Equivalent Decision Simplification: A New Method for Simplifying Algebraic Expressions in Genetic Programming

Mori, Naoki, McKay, R.I. (Bob), Nguyen, Xuan Hoai, and Essam, Daryl

***Abstract*— Symbolic Regression is one of the most important applications of Genetic Programming, but these applications suffer from one of the key issues in Genetic Programming, namely bloat – the uncontrolled growth of ineffective code segments, which do not contribute to the value of the function evolved, but complicate the evolutionary proces, and at minimum greatly increase the cost of evaluation. For a variety of reasons, reliable techniques to remove bloat are highly desirable – to simplify the solutions generated at the end of runs, so that there is some chance of understanding them, to permit systematic study of the evolution of the effective core of the genotype, or even to perform simplification of expressions during the course of a run.**

Previous approaches to arithmetic simplification rely on provably safe algebraic transformations, applied as rewrite rules, repeatedly re-writing segments of the expression until a minimal form is reached. This paper introduces an alternative approach, Equivalent Decision Simplification, in which subtrees are evaluated over the set of regression points; if the subtrees evaluate to the same values as known simple subtrees, they are replaced.

Equivalent Decision Simplification performs substantially better than standard approaches, generating far simpler expressions, but at the cost of computational time. It is thus well-suited to simplifying the final solutions generated by a run, or to post-run analysis, but is not intended to replace algebraic simplification in the course of a GP run.

I. Introduction

Genetic Programming (GP - [1]–[4]) has become well-known as a method for machine learning of models from data, generally for the purpose of predicting the values of previously un-seen data. In these applications, GP is used to generate models of the data, with the fitness criterion generally being to minimise some measure of the error in the data.

However GP suffers from a well-known problem, its propensity to generate large amounts of ineffective code (bloat – [2], [3], [5], [6]). Bloat causes a number of difficulties

1) Even though it has no effect on fitness, the ineffective code still has to be evaluated; as a result, the computational cost of GP is unnecessarily increased – in many cases, linearly in the amount of bloat
2) Bloat makes it difficult to understand the meaning of the model which has been generated, with the result that, while GP is frequently cited as a white-box learner (by contrast, for example, with neural networks), in reality it often functions as a black-box, yielding no useful explanatory power
3) Bloat hides the true complexity of the effective part of the model, making it difficult to trade off model complexity and accuracy, as is required under machine learning theory if GP is to yield models which generalise well to new data
4) Bloat masks the behaviour of the effective part of the code, making it difficult to discern or understand the evolutionary behaviour of the effective part of the code
5) Bloat masks the true structure of building blocks, making it difficult to determine the effect of an algorithm on building blocks (unless bloat is removed, two code segments with the same effective code are likely to be treated as different).

Bloat has been heavily researched, covering its causes, ways to avoid it, and ways to remove bloat (redundant code) from evolved trees. In this work, we emphasise simplification for the purpose of understanding the behavior of GP populations – for understanding the evolution of effective diversity, building blocks etc. For such analyses, the primary goal is complete – or at least, near complete – removal of ineffective code; computational efficiency is much less important. We contrast this with other applications, for example code simplification within GP runs in order to exert parsimony pressure, in which computational efficiency is crucial and completeness may be far less important.

In this paper, we first consider previous work on simplification – mainly algebraic simplification – in section II. Section III introduces Equivalent Decision Simplification, our new tree simplification method. The experimental context of this study is described in section IV, while section V provides the results of the simplification method, and some comparative results with algebraic simplification. Finally, in section VI, we discuss how these methods provide new information about the evolutionary behaviour of effective code GP systems, mention how the software may be accessed by other researchers, and discuss how we hope to extend this work in the future.

II. Background

A. Redundancy

In analysing GP dynamics, redundancy is a key issue. GP has redundancy in the genotype-to-phenotype mapping – that is, several individuals with different genotypes may nevertheless have the same phenotype. These different genotypes may have different complexities, and a GP algorithm is not constrained to find the simplest. As a result, GP can – and generally does – suffer from the phenomenon of bloat, in which both before and after the population has converged phenotypically, the complexity of the individuals increases rapidly.

We call the genotype components which typify bloat "redundant structures". We can categorise redundant structures into two main types:

- Neutral parts.
  If we change any node in a neutral part, it has no effect on the phenotypic value. For example, in $0 * f(x)$, $f(x)$ is a neutral part.

- Redundant expressions.
  We can represent a redundant expression by a different, smaller tree using some conversion, for example: $1 * f(x) \rightarrow f(x)$, where $1 * f(x)$ is a redundant expression.

In this study, we call a neutral part an "intron", and distinguish it from a redundant expression. By definition, trees which contain introns are redundant expressions – but there are also other types of redundant expressions.

Because of redundant structures, the situation can readily occur that the phenotype space is converged, but the genotype space has high diversity. It is well known that redundant structures are very important in maintaining robustness from crossover and mutation. However the effects of introns, redundant structures and effective code on GP search differ, so we need to distinguish them in analysis.

Regarding introns, Soule proposed the equivalent concept of "inviable code" [5], [6], and reported the size before and after removing the inviable code. However Soule did not study redundant expressions as a whole, but rather introns. Soule concluded that analysis of the amount of inviable code is very difficult, and proposed no general method to measure it [6].

Regarding redundant expressions, a very simple method using grammar rewrite rules was proposed by Koza [2], but no results were shown.

III. Simplification of GP Individuals

We call the operation of converting a tree structure into an equivalent but smaller structure "simplification". In this study, we propose some novel tree simplification methods.

First, we need to make the definition of simplification precise. Naturally, we would like tree simplification to minimise tree size under the condition that the trees' semantics are equivalent. We call such a simplification "canonical simplification". However, finding the canonical simplification of a given tree is very difficult. For example, if a tree represents a program, finding the canonical simplification is equivalent to finding the minimum description of that program (i.e. finding its Solomonoff/Kolmogorov [7] complexity). This is not Turing-computable. Therefore, we need to approximate. In this study, we compare two simplification methods, "rule-based simplification" (RBS), also known as "algebraic simplification" and "equivalent decision simplification" (EDS).

As mentioned in the introduction, we distinguish introns from redundant expressions. Figure 1 shows an example of an intron and a redundant expression.

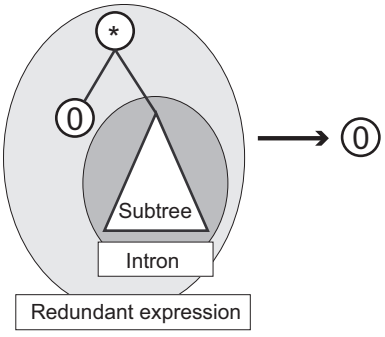


Fig. 1. Example of Intron and Redundancy Expression

A. Rule-based Simplification

For example, let the non-terminal nodes for a given problem be $\{+, -, *, /\}$, with terminal nodes $\{X, 1\}$. In this setting, canonical simplification would entail finding the minimal length formula obtainable by applying equivalent algebraic operations. However we can partially simplify by using rules such as, $1 * X \rightarrow X$ or $0 * X \rightarrow 0$. There are many such rules in arithmetic.

Soule's simplification [5], [6] may be viewed as rule-based simplification in which the simplification targets are limited to introns. However we have found rule-based simplification insufficient for our requirements. For example, the following formula is not readily simplified by rule-based simplification.

$$(X - 1) + (1 - X) \tag{1}$$

B. Equivalent Decision Simplification

We propose a new simplification method based on determining equivalence between specific simple trees and a subtree, known as "Equivalent Decision Simplification". In this context, "equivalence" depends on the problem domain; in a numeric domain such as symbolic regression, the determination of equivalence is made numerically. That is, if two expressions yield numerically equivalent values over a suitable range of inputs, they are regarded as equivalent. For example, equivalent decision simplification may be carried out as follows, in a symbolic regression problem.

1) Determine a suitable set of simple trees S_{simple}.
2) Check all subtrees in the target tree for equivalence to a tree in S_{simple}.
3) If some subtree is equivalent to a tree in S_{simple}, and larger than it, replace that subtree with the simple tree.
4) Repeat this procedure recursively until it fails.

In addition, we introduce an ordering on nodes, and sort the child nodes of commutative operators. One issue is how to determine a suitable S_{simple}. In symbolic regression problems, the set of terminal nodes – in our case, X, 0 (identity element of +) and 1 (identity element of *) – form a reasonable choice for S_{simple}, but of course more complex choices could reasonably be made.

EDS thus extends RBS by providing simplifications which are either:

1) too difficult to prove for use in RBS
2) true but unprovable in arithmetic (Goedel's Theorem)
3) not valid, but hold (or nearly so) for the instances used in training – in which case, they are equivalent from the perspective of the learning algorithm

From another perspective, we may think of RBS as a syntactically-based (proof-theoretic) simplification approach, and EDS as semantically-based (model-theoretic). Thus RBS simplifications correspond directly to proofs in the given domain, whereas EDS simplifications correspond to model-theoretic derivations of entailment. This has an important practical consequence: we may use the practical wisdom from many years of work in automated derivation to guide our choice of method: EDS is likely to be especially effective in domains where proof-theoretic methods are difficult (continuous arithmetic, random Boolean expressions), but may be comparatively more expensive in domains such as Horn clause

logic (logical rules) in which highly efficient proof-theoretic methods are known.

Despite its computational cost, even equivalent decision simplification is not a universal panacea. For example, we cannot simplify $(X+1)(X-1)$ to X^2-1 unless we include X^2-1 in S_{simple}.

IV. Experimental Setting

In this study, we use standard GP and focus on a continuous real-arithmetic domain, using a typical symbolic regression problem [2]–[4], [8].

A. Problem Domain

The chosen problem is, given the 20 random X and Y values shown in Figure (2) over the range $[-\pi, \pi]$, to find an expression for the target function $\cos 2X$ (in the figure, the 20 points are represented by + symbols). The 20 points are generated by dividing the range into 20 even intervals, and sampling uniformly randomly across each interval. The sample is generated once for each run, then held constant throughout the run.

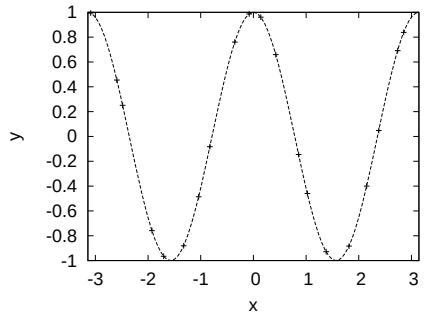


Fig. 2. 20 sample points

Problem

Objective function : $\cos 2X \quad [-\pi, \pi]$
Operators : $\{+, -, *, \%, \sin\}$
Operands : $\{X, 1\}$

where "%" is protected division, satisfying $X\%0 \rightarrow 1$. (note that the target function, cos, is not included in the operator set).

This problem has three separate simple solutions:

opt1 $1 - 2\sin^2 X$
opt2 $\sin(\frac{\pi}{2} - 2X)$
opt3 $\sin(\frac{\pi}{2} + 2X)$

Of course, there are others: any normalised linear combination, and any transformation of sine's argument by $n * 2\pi$, will give a new solution, so that there are infinitely many different solutions

B. GP Settings

The GP settings used in our experiments are as follows:

- Number of runs: 1000
- Generations per run: 200
- Population size: 500
- Crossover rate: 0.9, using subtree crossover
- Mutation: subtree mutation, rate 0.1
- Selection: tournament selection, tournament size 3
- Initialisation: ramped half-and-half
- Tree Depth limit: initial limit = 6; subsequent limit = 15
- Repair mechanism: on depth violations, re-try up to 100 times

The raw fitness(RF) is calculated from the sum of the absolute errors at the 20 data points. Given 20 fitness points $S_{\text{t}} = \{X_i, i = 1, 2, \ldots, 20\}$, the fitness f of the individual which represents function $g(X)$ is given by:

$$E = \sum_{i=1}^{20} | \cos 2X_i - g(X_i) | \quad (2)$$

$$f = \frac{1}{1+E} \quad (3)$$

An individual is regarded as a solution when all 20 errors are less than 0.01, as follows:

$$g(X) = \begin{cases} \text{solution}, & \forall X_i \in S_{\text{t}}, \\ & | \cos 2X_i - g(X_i) | \leq 0.01 \\ \text{non} - \text{solution}, & \text{otherwise} \end{cases} \quad (4)$$

Table I shows the rules used in rule-based simplification.

TABLE I
Rewrite rules

$A+0 \rightarrow A$, $0+A \rightarrow A$, $A \times 1 \rightarrow A$, $1 \times A \rightarrow A$
$A \times 0 \rightarrow 0$, $0 \times A \rightarrow 0$, $\sin 0 \rightarrow 0$
$X - X \rightarrow 0$, $1 - 1 \rightarrow 0$, $A - 0 \rightarrow A$
$0\%0 \rightarrow 1$, $A\%0 \rightarrow 1$, $0\%A \rightarrow 0$, $X\%X \rightarrow 1$, $A\%1 \rightarrow A$

where A represents any subtree, and % represents protected divide.

S_{simple} in III-B is set to $\{0, 1, X\}$.
The flow of simplification is as follows:

1) Let the genotype tree of individual i be t_i
2) Apply rule-based simplification recursively to all nodes of t_i, until there is no node to which rule-based simplification can be applied, obtaining t_i'.
3) Apply equivalent decision simplification to all nodes of t_i'. If any node is translated, let the translated tree be t_i and go to (2). If there is no node to which EDS can be applied, finish and let t_i' be the final result.

That is, simplification is carried out repeatedly until neither type of simplification – rule-based or equivalent decision – can be applied.

C. Numerical Computation Issues

This work leads to some difficult issues in numerical computation, which we consider together here.

1) Computing Zero: Because of numerical computation errors, the issue of exactly what values should be regarded as "Zero" is complex. For example, the value of the formula

$$\frac{1}{\sin(1)} \times \sin(1) - 1 \quad (5)$$

is theoretically zero. However in code generated by the g++ compiler on an intel 686-class architecture, this expression evaluates to -1.0625E-17. One "solution" is to ignore this. However it seems undesirable, mainly because of the effect on protected division. For example, X $\%\epsilon$, with $\epsilon \ll 1$ (e.g. ϵ might arise from eq. 5). If ϵ is treated as 0, it evaluates to 1; if ϵ is treated as nonzero, it evaluates to a very large number. We don't believe there is any completely satisfactory general solution. Pragmatically, we chose to:

- Treat values below 10^{-9} as 0.
  The choice of 10^{-9} was somewhat arbitrary; however we note that in this problem, the criterion for a solution uses 0.01, and the range of x is $[-\pi, \pi]$, leaving a large margin.
- Use the same criterion in both the GP runs and the analysis program.

We note that, comparing otherwise identical runs with different values of ϵ in the 10^{-9} range, differences arise after about generation 20, so the choice of ϵ does have a perceptible effect.

2) Computing Phenotypic Measures: In this paper, the "phenotype" is the numerical vector of the 20 function values of an individual, calculated at the 20 sample points. In order to calculate phenotypic quantities such as the total number of phenotype values or the entropy, we need to compare phenotype vectors, testing whether they are the same. We could not find a completely satisfactory theoretical definition for 'the same'; pragmatically, we chose to:

- Set the acceptable error value $\epsilon_{\rm p}$.
- Distinguish phenotypic vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ by calculating the relative error E_i of each of their components i (if v_{1i} is zero, then v_{2i} must also be zero; in this case, we may skip the corresponding relative error check):

$$E_i = \frac{|v_{1i} - v_{2i}|}{|v_{1i}|} \tag{6}$$

  (we use the relative error rather than the absolute error to avoid scale effects)
- If, for all i, either $E_i < \epsilon_{\rm p}$, or both v_{1i} and v_{2i} are zero, then treat $\mathbf{v}_1$ and $\mathbf{v}_2$ as identical.

How should we choose this $\epsilon_{\rm p}$ value? In this work, we chose to set $\epsilon_{\rm p}$ to a value sufficient to distinguish between a solution and a nearby non-solution. That is, we wish to distinguish between a solution vector $\mathbf{v}_{\rm sol}$ and a neighbouring non-solution vector $\mathbf{v}_{\rm non}$ represented by:

$$\mathbf{v}_{\rm sol} = \{\cos 2x_i\}, \quad x_i \in S_{\rm T} \tag{7}$$

$$\mathbf{v}_{\rm non} = \{\cos 2x_i\} + \{\delta, 0, 0, \cdots, 0\}, \quad x_i \in S_{\rm T} \tag{8}$$

(note that the position of δ in the tuple has no effect).

For δ, we use the error value 0.01 previously chosen for checking solutions. Then the relative norm error between $\mathbf{v}_{\rm opt}$ and $\mathbf{v}_{\rm non}$ may be defined as:

$$\frac{||\mathbf{v}_{\rm sol} - \mathbf{v}_{\rm non}||}{||\mathbf{v}_{\rm sol}||} = \frac{\sqrt{\delta * \delta}}{||\mathbf{v}_{\rm opt}||} \tag{9}$$

$$= \frac{0.01}{3.148273830304322...} \tag{10}$$

$$= 0.0031763437804370942... \tag{11}$$

$$\equiv \epsilon_{\rm pheno} \tag{12}$$

(where 3.148273830304322 was calculated across the data points in our simulations). We then use $0.9\epsilon_{\rm pheno} \approx 0.00286$ as the critical error for distinguishing two vectors (0.9 is a margin to avoid errors). That is, we use, $\epsilon_{\rm p} = 0.0028587094023933847...$. Note that it is not possible to simply use g++'s `==` for comparison, because each element of the vector is represented as a `double`.

We used this to calculate phenotypic entropy $H_{\rm p}$ as follows:

1) Let N_i be the number of individuals in population P which belong to phenotype i.
2) We treat the ratio ($N_i/N_{\rm P}$) as an estimate of the probability p_i of i. ($N_{\rm P}$: population size)
3) We compute the phenotypic entropy $H_{\rm p}$ as:

$$H_{\rm p} = -\sum_{i \in P} p_i \log_e p_i \tag{13}$$

3) Computing Genotype Entropy: Given a population of genotypes (simplified or not), computation of the genotype entropy $H_{\rm g}$ parallels that of the phenotype entropy:

1) Let N_g be the number of instances in population P of genotype g
2) As before, we treat the ratio ($N_g/N_{\rm P}$) as an estimate of the probability p_g of g.
3) We compute the genotypic entropy $H_{\rm g}$ as:

$$H_{\rm g} = -\sum_{g \in P} p_g \log_e p_g \tag{14}$$

4) Computing Simplifications: In equivalent decision simplification, we need to check whether each subtree is equivalent to X, "1" or "0". We do this by checking whether it yields the same value on each of the 20 sample points $S_{\rm T}$. But again, we have to deal with the problem of numerical error – what does "the same value" mean? Again, we take a pragmatic approach, as there does not seem to be much theory to fall back on. Let $f_s(X)$ be the function computed by subtree s. To check whether it is equivalent to "0", the criterion mentioned in sub-section IV-C.1 is used (the 20 values of $f_s(X)$ must be below 10^{-9}). For X and 1, we check if the 20 corresponding errors between $f_s(X)$ and $\{X \text{ or } 1\}$ are below 10^{-3} (0.1%). If so, $f_s(X)$ is regarded as equivalent to $\{X \text{ or } 1\}$. The following is pseudocode for the test for X.

```
/*
  STest    array of 20 sample data
  fs       function computed by subtree
  fx       function f(x)=x
*/
for x in (STest){
    error = relativeError(fx, fs(x));
    if(error > 0.001) return false;
}
return true;
```

There are many cases this criterion is not able to check. For example, consider an expression such as: $a + a$, where $a = 0.9 \times 10^{-9}$. If we test whether the whole expression from node "+" is zero or not, the test will fail since the value is $2 \times 0.9 \times 10^{-9} = 1.8 \times 10^{-9} > 10^{-9}$. However, a will be found equivalent to zero because the value is below 10^{-9}. At the next step, $a + a \rightarrow 0 + 0 \rightarrow 0$. Of course, this case is not damaging, but there is some risk in other cases of growing errors to large values. How to treat this problem is deserving of further study.

D. Experimental Trials

1000 independent runs of the GP system were performed, using 1000 different random seeds, the entire populations being saved for the subsequent analyses.

Part of this investigation focuses on illuminating the differences between runs which are rapidly successful in finding a solution, runs which are successful more slowly, and runs

which do not succeed in finding a solution. To this end, three separate sets of 100 runs were selected from the 1000-run sample by stratified random sampling:

- C_{20}, consisting of 100 runs which found a solution between generations $20 \sim 29$.
- C_{50}, consisting of 100 runs which found a solution between generations $50 \sim 69$.
- C_{fail}, consisting of 100 runs which did not find a solution within the 200-generation limit of the runs.

V. Results and Discussion

In this section, we show a number of graphs and tables describing the results of our experiments and analyses. In all plots, the x axis shows the generation, while the y axis shows the value of the particular parameter under investigation. The plots show three graphs, corresponding to C_{20}, C_{50} and C_{fail} (i.e. fast-solving, slow-solving and failing runs – respectively opt 20-29, opt 50-69 and fail in the legends).

It is important to consider what EDS buys us over RBS. Let us take as an example the 100 C_{20} runs. These found a total of 102,534 different genotypes which, under EDS, reduced to size 12 (i.e. one of the three solutions noted in table V). The average size of these genotypes (prior to simplification) was 138.2. Thus more than 90% of the structure of these solutions was redundant code.

On the other hand, if we apply only RBS, the average size of the solutions after simplification is 15.1 – that is, around 25% of EDS simplifications are missed. In fact, only about 39% of the genotypes are simplified to size 12. At the other extreme, 57 of these genotypes – which EDS simplifies to 12 nodes – retain more than 100 nodes, even after rule-based simplification; in the worst cases, two (apparently coincidentally) have 214 nodes. These are presented as examples in table II[1]

The rule-based simplification reported here is strictly more powerful than previous studies [5], [6], since it simplifies some instances of redundant expressions in addition to the introns removed in the cited works. We thus argue that our simplification approach (combining rule-based and equivalent decision methods) constitutes a significant improvement, permitting more reliable analyses of effective code. The rest of this section outlines some kinds of analyses we can perform because of the effectiveness of EDS.

A. Evolutionary Dynamics of Genetic Programming

In this subsection, we consider the evolutionary dynamics of Genetic Programming; for economy of exposition, we use the plots of C_{20}, C_{50} and C_{fail} rather than the overall population plots, but pay attention mainly to the overall shapes of the graphs, rather than the differences between them.

Typical studies into GP evolutionary dynamics generate plots such as figures 4(a) (showing the size of trees in the evolving populations), and 3(a) and (c) (showing the genotypic and phenotypic entropies) – though the very limited information provided by the genotypic entropy in this form means that it would usually be ignored.

From these information sources, what can we observe? Firstly, figure 4(a) shows that the un-simplified tree size is monotonically increasing, but little else.

[1]Expressions are presented in Reverse Polish Notation; "S" represents sine, and "/" protected division.

The genotypic diversity metric is fairly uninformative. It starts at a value just above 6, rapidly increases to around 6.16 (the theoretical maximum is $\log_e 500 \approx 6.21$), then from about generation 10 falls very gradually to around 6.14.

The phenotypic diversity metric is perhaps more interesting. Figure 3(c) certainly shows some structure. It starts at a value just below 6 . It falls rapidly to a local minimum around 4.5 (generation 4), rises rapidly to around 5.4 (generation 10), then falls monotonically toward an asymptote.

The initial rapid loss of diversity is entirely expected, as the system rapidly eliminates highly unfit individuals. The next phase, of rapidly increasing phenotypic entropy, is slightly surprising, but we will see below that our new methods are able to provide at least a partial explanation, in terms of increasing size permitting an increasing range of available effective code genotypes, and hence allowing the variation operators to dominate selection, at least for a time. Finally, though, sufficiently good solutions are found that selection dominates and diversity decreases.

EDS allows us to generate two further figures, 4(b) and 3(b). Interestingly, the latter appears to bring us little advantage – it appears very similar to figure 3(c). This is an important point – the phenotype entropy and the genotype entropy of subtrees simplified by EDS are almost identical, strongly suggesting that EDS has found most available simplifications. This is confirmed by statistical testing. The differences between figures 3(b) and (c) in their C_{20}, C_{50} and C_{fail} values respectively, were tested using Welch's two-sided t-test [9]. At no point in the evolution of any of these three sets of trials did the difference between the effective genotype entropy and the phenotype entropy reach even the 5% significance level. This suggests that EDS is doing an effective job, in that there is high similarity between the number of genotype species and the number of phenotype species. That is, if two expressions are in fact equivalent (in the sense of generating the same values at the 20 sample points), then their reduced genotypes, after EDS, are likely to be identical – i.e. EDAS has found the true minimal form in most cases.

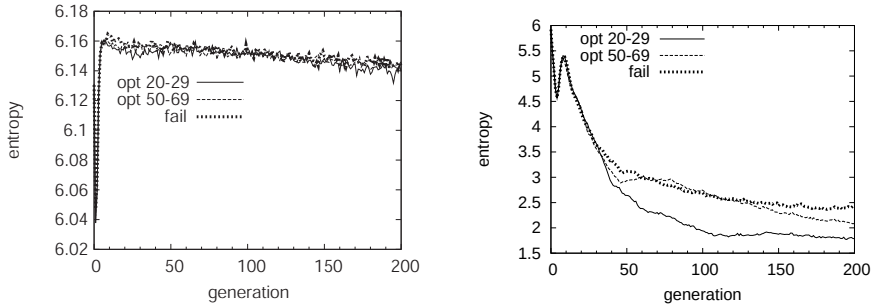
Figure 4(b) reveals a great deal of additional information about the runs. Firstly, unlike the total node size, the effective node size drops sharply in the first few generations, confirming our earlier conclusion that many solutions are rapidly lost, and suggesting that many of the individuals lost have relatively large effective code size. The effective code size then rises rapidly, as with the phenotype entropy, until about generation 10, when there is a local maximum. We interpret this phase as the discovery of some relatively fit, small expressions such as X, 1 and $\sin X$, These small expressions out-compete the first successful group of larger individuals, and size declines. Interestingly, this effect seems to be stronger in runs which are destined to succeed soon thereafter, suggesting that at least some of these small expressions – perhaps $\sin X$ – are useful building blocks.

We note that the effective-code-size behaves in a slightly different way than we might expect from the seminal papers of Nordin and Banzhaf on parsimony pressures on effective code in [10]. After an initial drop in size subsequent to finding the solution, effective code size continues to rise. However our problem may be slightly atypical in this respect, since the need for accurate approximations to π may counterbalance the effect

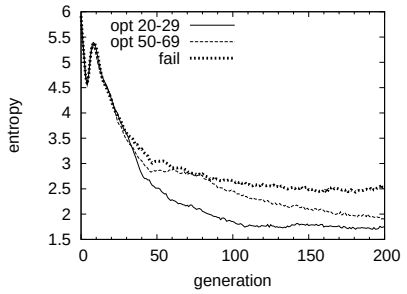
TABLE II

SIMPLIFICATION EXAMPLES: RULE BASED VS EQUIVALENT DECISION SIMPLIFICATION[1]

	Expression 1	**Expression 2**
Original	276 Nodes: `111++1-1X/1X/X-1-1X/+1*1S1*/1S1*/+11+11*X 1X*1-X*XX+X1/X*-++X11-1+1X-//+*/-*X1111*- --S1*+1SX1/1+1X-1X-*X1111*---S1**/-X1S1*X *1-11*11++1+S*/1+X1-S1X-*1S1X-*S1X-X**11- 1*X/+*/1X/+1S1XX11++1-//1+1S1X-*1SS111S+* *1SX1-X+-1*X/+*/-+*+XX1X++1-/11+*1-*11/11 +SS*X+XX1X++1-/11+*1111-/*+*+S`	294 Nodes: `X1X++1-111+/XX/X-X1X++-1X/+XXX*1X-/*S*1S1 */XX+1*1X*-XX*X+XX+1/+*XX1*1X+/*1X1X-*-*- 1*/+11+11*XX1-11+/X*X1-X+X1/X*-++111-1+X/ /+*/-*X1111*---S1*+1S11/1+11X-*1X+*/-11*1 +X+SX1/11*1+/1X*1-11X*/11/+X+*/+1S11/1+1S 1X-*XX1//SS111S+**11-1*X/+*/-+*+XX1X++1-/ 11+*1-*11/11111*--+SS*X+XX1X++1-/11+*1111 -/*+*+S`
Simplified by RBS	214 Nodes: `X1SX*1-11+1+1+S*/1+1X-X1-S*1X-X*1X-1S*S** /1X/+1S1X11+X+1-//1+1X-1S*1S1+1SS*1SX1-X+ -X/+*/-+1S1X+1X-1X-*X1-S*/-*11+1+1-1X/1X/ X-1-1X/+1S/1S/+11+X1-X*XX+XX*-+X+X11X-//+ /-*X1-S++11+X1X+X+1-/*1-*11+SSX+11+X1X+X+ 1-/*1+*+S`	214 Nodes: `X11+/1X/1+X+X1-*/11+X+S+1S11+1X-1S*1S1+1S S**/-+1S11+1X+1X-*/-*1X+X+1-111+/1X-1X+X+ -1X/+XX*1X-/X*S*1S/XX*X+XX++XX+X-*X1X+/X* 11X-X*-*-/+11+X1-11+/X*X1-X+XX*-+X+11X//+ /-*X1-S++11+X1X+X+1-/*1-*11+SSX+11+X1X+X+ 1-/*1+*+S`
Simplified by EDS	12 Nodes: `11+SSX+11+*S`	12 Nodes: `11+SSX+11+*S`



(a) Genotype Entropy (before simplification) (b) Genotype Entropy (after simplification)



(c) Phenotype Entropy

Fig. 3. Variation in Entropy of Genotype and Phenotype

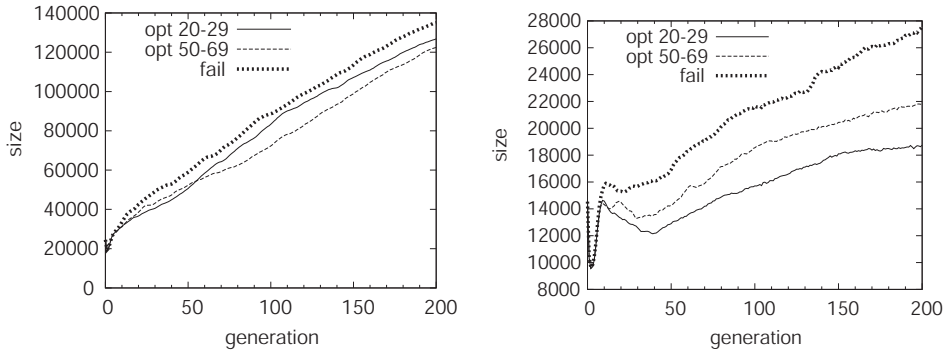
of parsimony pressures arising from destructive crossover.

B. Comparisons of Successful and Unsuccessful Runs

When we come to examine the differences between successful and unsuccessful runs, we note first that figure 4(a) confirms the results of [11], [12], who were unable to detect a relationship between size, and when, or whether, a solution was found. Of course, it goes almost without saying that the un-simplified genotype entropy (figure 3(a)) is also unable to make such a distinction.

However the phenotype entropy (figure 3(c)) *is* able to distinguish the successful from the unsuccessful runs, albeit only retrospectively. The phenotype entropy of a successful run is lower than that of an unsuccessful run in the generations after the solution has been found, the asymptotes being around 1.7 for a successful run, 2.5 for an unsuccessful run.

The size of the effective code (figure 4(b)) is highly correlated with the success of evolution, the three different classes, C_{20}, C_{50} and C_{fail}, being readily distinguished. They first become distinct around generation 15 – well before the first solutions are found – indicating that effective code size is able to *prospectively* identify successful runs. Thus EDS provides important new information, not available without an effective



(a) Size (before simplification) (b) Size (after simplification)

Fig. 4. Variation in the Total Number of Nodes (Size)

simplification procedure.

C. The Effects of Evolution on Frequent Small Subtrees

TABLE III
MOST FREQUENT 3A TEMPLATES IN UN-SIMPLIFIED INDIVIDUALS[1]

Generation:	80	120	200
C_{20}	$11+$, $11-$	$11+$, $11-$	$X1-$, $11+$
C_{50}	$11-$, $XX-$	$11-$, $11+$	$1X-$, $11+$
C_{fail}	$XX-$, $11-$	$XX-$, $X1-$	$1X-$, $X1-$

In this sub-section, we consider the effects of evolution on the frequencies of small subgraphs. We studied a number of different shapes, but here we report on one, perhaps the most informative, consisting of 3 nodes, a parent binary node and its two sub-child nodes. This graph we call template 3a to distinguish it from the other possible subgraphs of size 3 (which are all linear graph shapes). Table III shows the two commonest template 3a type subtrees in generation 80, 120, 200 of C_{20}, C_{50} and C_{fail} in the un-simplified trees. For example, $11+$ is the commonest, and $11-$ the second most common, template 3a subtree in un-simplified trees of runs that found a solution between generations 20 and 29 (i.e. C_{20}) at generation 80.

TABLE IV
MOST FREQUENT 3A TEMPLATES IN EFFECTIVE CODE OF INDIVIDUALS[1]

Generation	80	120	200
C_{20}	$11+$, $S1+$	$11+$, $S1+$	$11+$, $S1+$
C_{50}	$11+$, $XX+$	$11+$, $S1+$	$11+$, $S1+$
C_{fail}	$1X+$, $11+$	$1X+$, $11+$	$1X+$, $11+$

Table IV shows the equivalent analysis for effective code. (Note that, unlike table III, some of the expressions are not terminal expressions, so $sin(...)$, for example, is internal, and may be completed in different ways).

The first point to note from table III is that expressions representing zero ($11-$, $XX-$ etc.) are very common; in generation 80, only in the C_{20} runs does a non-zero subtree ($11+$) appear. By generation 120, both C_{20} and C_{50} feature ($11+$), while C_{fail} shows a different nonzero expression, $X1-$. By generation 200, zero-equivalent code has disappeared from the top two places in all three runs. Since the nonzero expressions differ somewhat from those which are found commonly in the effective code (table IV), it is difficult to explain their frequency as a result of selection; a full understanding of this phenomenon requires further study.

Conversely, table IV shows that $11+$ is common in the effective code by generation 80, in all three cases – (C_{20}, C_{50} and C_{fail}). However $\sin$ appears by generation 80 only in C_{20}. By generation 120, $11+$ and $S1+$ are the two commonest templates in C_{20} and C_{50}, yet $\sin$ has not yet appeared in C_{fail}, and indeed the result is unchanged even by generation 200.

In our experiments, no solutions of the $1-2\sin^2 X$ family were found, all solutions being from the $\sin(\frac{\pi}{2} \pm 2X)$ family. To build this type of solution requires only one occurrence of $XX+$, the more complex problem being to build an approximation to $\frac{\pi}{2} + 2n\pi,\ (n = 0, \pm1, \pm2, \cdots)$. We suspect that $S1+$ is important in building this term, explaining its predominance in C_{20} and C_{50} runs, and perhaps explaining why the runs in which it was less common – the C_{fail} runs – failed to find a solution.

It is worth noting that this analysis is only available to us because of table IV; it is not apparent from table III. That is, a potent simplification method is necessary to understand the importance of this code fragment, and potential building block, to finding a solution.

D. The Solutions Found

TABLE V
EFFECTIVE CODE OF SMALL SOLUTIONS[1]

Genotype	size
$11+11+SSX+*S$	12
$11+11+SSX-*S$	12
$11+SSX+11+*S$	12
$XX+1SS1S1-/+S$	13

We investigate which classes of solutions are found by evolution. Table V shows the two smallest (measured in effective code) solutions found in C_{20} and C_{50} runs, of sizes 12 and 13.

The size 12 solutions can all be represented generally as:

$$\sin(2\sin(\sin 2) \pm 2X) \approx \sin(\frac{\pi}{2} \pm 2X) \qquad (15)$$

That is, GP solved the problem using the approximation

$$2\sin(\sin 2) \approx 1.5781 \approx \frac{\pi}{2} \qquad (16)$$

This, in itself, is interesting - that a sufficiently good approximation to $\pi/2$ can be formed (and found by GP) with only 8 nodes. However the size 13 solution is perhaps even more interesting, having the general form

$$\sin\left(2X + \frac{\sin(\sin 1)}{\sin 1 - 1}\right) \approx \sin\left(\frac{\pi}{2} + 2X - 2\pi\right) \qquad (17)$$

and using the approximation

$$\frac{\sin(\sin 1)}{\sin 1 - 1} \approx -4.7034 \approx -\frac{3}{2}\pi \qquad (18)$$

In other words, GP has been able to find a solution approximating not the first, but the second member of the family of solutions, using only 9 nodes for the numeric approximation. It is worth noting that phenotypic analysis, or indeed genotypic analysis without strong simplification, could never tell us about this.

VI. Conclusions

We have proposed a novel GP simplification method, equivalent decision simplification. We showed that EDS could achieve substantially greater simplification than previous rule-based methods. Applying this method, we were able to obtain the following:

- An analysis of the quantitative dynamics of problem complexity (bloat).
- An understanding of a useful correlation between size and solution-finding.
- An understanding of the roles of common subtrees in the success/failure of particular searches.
- An analysis of the different classes of solutions found by GP search.

In other work [13], we have applied EDS and compression analyses to understand the evolution of regular genotype structure in a range of different GP systems – briefly, we found that standard evolutionary systems did not evolve regular structure; developmental systems could initially generate regularity, but without additional mechanisms (developmental evaluation) could not maintain it; and that this effect was particularly marked in the effective code, as opposed to the overall code.

The EDS method presented here has been implemented as an extensible series of class libraries. The libraries are available from http://sc.snu.ac.kr (click on the 'software' tab).

We plan to extend this work by applying EDS to a range of other symbolic regression problems, and to investigate the extension of EDS to non-arithmetic problem domains

Acknowledgment

The authors would like to acknowledge helpful discussions with Prof. Hajime Kita of Kyoto University, Japan, and assistance with preliminary experimental work from Mr Tuan Hao Hoang, of the University of New South Wales, Australia. They would particularly like to thank Mr Seiji Morimoto, who made major contributions to programming the public version of the simplification library mentioned here.

We especially wish to acknowledge the unfailing support and encouragement of Prof. Keinosuke Matsumoto of Osaka Prefecture University, Japan.

This research was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Young Scientists (B), 18700227, 2006-2007, and by the support program for new faculty of Seoul National University.

References

[1] N. L. Cramer, "A representation for the adaptive generation of simple sequential programs," in *Proceedings of an International Conference on Genetic Algorithms and the Applications*, J. J. Grefenstette, Ed., Carnegie-Mellon University, Pittsburgh, PA, USA, 24-26 July 1985, pp. 183–187. [Online]. Available: http://www.sover.net/ nichael/nlc-publications/icga85/index.html

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

[3] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, dpunkt.verlag, Jan. 1998.

[4] J. R. Koza, M. A. Keane, M. J. Streeter, W. Mydlowec, J. Yu, and G. Lanza, *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers, 2003. [Online]. Available: http://www.genetic-programming.org/gpbook4toc.html

[5] T. Soule and J. A. Foster, "Support for multiple causes of code growth in GP," 20 July 1997, position paper at the Workshop on Evolutionary Computation with Variable Size Representation at ICGA-97.

[6] T. Soule, "Code growth in genetic programming," Ph.D. dissertation, University of Idaho, Moscow, Idaho, USA, 15 May 1998. [Online]. Available: ftp://ftp.cs.uidaho.edu/pub/foster/papers/soule-thesis.ps.gz

[7] R. Solomonoff, "A theory of inductive inference," *Information and Control*, vol. 7, pp. 1–22, 224–254, 1964.

[8] N. X. Hoai, "Solving trignometric identities with tree adjunct grammar guided genetic programming," in *2001 International Workshop on Hybrid Intelligent Systems*, ser. LNCS, A. Abraham and M. Koppen, Eds. Adelaide, Australia: Springer-Verlag, 11-12 Dec. 2001, pp. 339–352.

[9] B. Welch, "The significance of the difference between two means when the population variances are unequal," *Biometrika*, vol. 29, pp. 350–362, 1938.

[10] P. Nordin and W. Banzhaf, "Complexity compression and evolution," in *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, L. Eshelman, Ed. Pittsburgh, PA, USA: Morgan Kaufmann, 15-19 July 1995, pp. 310–317. [Online]. Available: ftp://lumpi.informatik.uni-dortmund.de/pub/biocomp/papers/icga95-1.ps.gz

[11] E. Burke, S. Gustafson, and G. Kendall, "A survey and analysis of diversity measures in genetic programming," in *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, Eds. New York: Morgan Kaufmann Publishers, 9-13 July 2002, pp. 716–723.

[12] S. Gustafson, A. Ekart, E. Burke, and G. Kendall, "Problem difficulty and code growth in genetic programming," *Genetic Programming and Evolvable Machines*, vol. 5, no. 3, pp. 271–290, Sept. 2004. [Online]. Available: http://www.cs.nott.ac.uk/ smg/

[13] J. Shin, M. Kang, R. I. B. McKay, X. Nguyen, T. H. Hoan g, N. Mori, and D. Essam, "Analysing the regularity of genomes using compression and expression simplification," in *Proceedings of the 10th European Conference on Genetic Programm ing (EuroGP2007, Valencia, Spain)*, ser. Springer Lecture Notes in Computer Science. Berlin, Germany: Springer-Verlag, April 2007, vol. 4445, pp. 251–260.