Canonical Form Functions as a Simple Means for Genetic Programming to Evolve Human-Interpretable Functions

Trent McConaghy
K.U. Leuven
Kasteelpark Arenberg 10
B-3001 Leuven, Belgium
Trent.McConaghy
@esat.kuleuven.be

Georges Gielen
K.U. Leuven
asteelpark Arenberg 10
B-3001 Leuven, Belgium
Georges.Gielen
@esat.kuleuven.be

ABSTRACT

In this paper, we investigate the use of *canonical form functions* to evolve *human-interpretable* expressions for symbolic regression problems. The approach is simple to apply, being mostly a grammar that fits into any grammar-based Genetic Programming (GP) system. We demonstrate the approach, dubbed CAFFEINE, in producing highly predictive, interpretable expressions for six circuit modeling problems. We investigate variations of CAFFEINE, including Grammatical Evolution vs. Whigham-style, grammar-defined introns, and smooth uniform crossover with smooth point mutation (SUX/SM). The fastest CAFFEINE variant, SUX/SM, is only moderately slower than non-grammatical GP – a reasonable price to pay when the user wants immediately interpretable results.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search; I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms

Keywords

Genetic programming, grammar, grammatical evolution

1. INTRODUCTION

The *prediction* problem encompasses many real-world applications including financial forecasting and weather prediction. Because of the value, there is extensive research to find better approaches, e.g. with splines [7], neural networks [27], support vector machines [6], and of course genetic programming (GP) in symbolic regression (SR) mode [14]. For prediction, GP sits in a competitive, crowded field.

In contrast to prediction, in the problem of *knowledge extraction*, one aims to get *insights* from a dataset. This might be as simple as visualizing raw data, but it is potentially far more valuable is to first process the data in some way. For example, one might apply a dimensionality-reduction algorithm then view transformed data in three dimensions. Or, one might use a tool to compute the relative impact of different variables on a target variable. Even better would be to find insights in the relations among variables, e.g. by visualizing CART trees [2] or extracting rules from neural networks [29]. Ideally, one could find a *functional relation* from some variables to other variables, i.e. find a "whitebox" mapping from a dataset. Though non-GP approaches to whitebox modeling exist, (e.g. [16]), the searching ability and flexibility of GP in SR mode make it ideally suited to the task.

Or at least, that's what GP *should* be suited for. But as a recent paper complains, "[GP-evolved] expressions can get, as we have seen, quite complex, and it is often extremely difficult to understand them without a fair bit of interaction with a tool such as *Mathematica*" [13]. Fortunately, we can identify specific interpretability challenges: managing complexity, excessive compounding of nonlinear operators, finding good coefficient values, simultaneous overabundance and shortage of coefficients, non-compact polynomials and rationals, dimensional awareness, bounded ranges for expressions, and bounded ranges for operators.

Some GP systems deal with these challenges quite well, most notably adaptive logic programming [10]. Unfortunately, its adoption is hindered because it is substantially more complex than off-the-shelf (grammar-based) GP systems.

In this paper we present an approach that addresses most of the above challenges as well. The key idea is to use CAnonical Form Function Expressions IN Evolution (i.e. CAFFEINE). This makes it far easier for GP to place coefficients and manage complexity and nonlinearity. It is simple to apply, as it is mostly a grammar that fits into any grammar-based GP system. The method will be illustrated for analog circuit performance modeling.

This paper is organized as follows. In section 2, we review SR challenges, and related approaches. In section 3 we describe CAFFEINE. Section 4 demonstrates its application to knowledge extraction in analog design. Section 5 investigates the relative performance of variants, including Grammatical Evolution vs. Whigham-style, grammar-defined introns, and smooth uniform crossover with smooth point mutation. Section 6 concludes.

2. CHALLENGES OF SR

This section examines SR challenges and GP approaches (or lack of approaches) to handle each. Most of them are specific to SR.

Managing Complexity. Occam's Razor is the guide here: the simplest model that describes the data is usually the correct one. Complexity is typically dependent on measures like tree depth and node count. In GP, expression-simplification processes are of two varieties: non-SR and SR-specific. Non-SR techniques include (1) penalizing complex solutions ("parsimony pressure"), (2) having complexity as a second objective and using a multiobjective algorithm, (3) maximum tree depth [14], (4) uniform operators such that depths never grow [26], and (5) other "bloat control" methods, e.g. [25]. The SR-specific approach is to do symbolic simplification, either automatically during or after evolution with a symbolic math tool like *Mathematica*, or manually after evolution.

Excessive Compounding of Nonlinear Operators. GP gives uniform treatment to operators, whether they are linear or nonlinear. The result is that even a very small tree which would pass GP-parsimony standards could be not interpretable by humans. An example is tan(exp(sin(x))): three compounded nonlinear operators is too much, and even two is questionable. Maximum tree depth might handle this, but unfortunately it has to still be large enough to handle other reasonable combinations of expressions such as polynomials.

Finding Coefficient Values. Induced expressions might have real-valued coefficients which must be determined during GP search. Coefficients can be either the linear "weights" on each basis function (along with the offset), or the nonlinear coefficients inside basis functions. Linear weights can be handled by: inclusion with nonlinear coefficients; linear regression [20]; or using just one basis function and a simple correlation calculation to sidestep linear regression until after evolution [12]. Nonlinear coefficients can be handled by "ephemeral random constants" [14]; by constant perturbation on a distribution, e.g. uniform [28] or Gaussian [1]; via nonlinear optimization [30]; within a constant-creation grammar such as digit concatenation [5]; or even combining multiple strategies [5].

Log-Range of Coefficient Values. For some problems, coefficient values should also be able to take on a wide range of possible values that may vary by many orders of magnitude: large positive numbers like 1.0e+10, small positive numbers like 1.0e-10, zero, small negative numbers, and big negative numbers. Some SR approaches handle it implicitly by allowing log() and/or power() operators to act directly on the constants, or by choosing from a discrete set of log-range variables. We have not been able to identify an work that directly addresses log-valued constants for continuous-valued numbers.

Coefficient values are just one side of the "coefficient coin"; GP must also determine *where* in the expression to insert each constant. Thus, in contrast to much research on coefficient values, with the exception of the linear/nonlinear distinction, to our knowledge there is little discussion of coefficient placement in the GP literature. Unfortunately, this means that GP-evolved equations can end up having too few constants in some places and too many in others; i.e. *shortages* and *overabundances*.

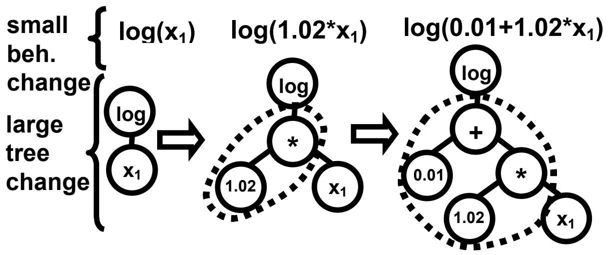

Figure 1: Coefficients can be difficult to insert if not already present, even if the behavioral change is small

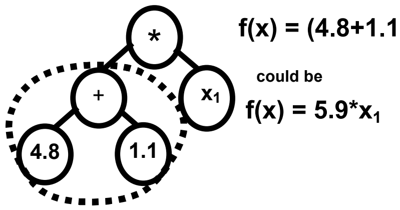

Figure 2: An example of coefficient overabundance

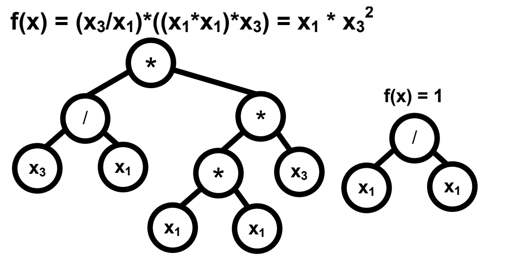

Figure 3: It can take many nodes to get a simple polynomial or rational expression. They can even cancel each other out.

Coefficient Shortages. Consider the expression f(x) = log(x), which might appear in a typical SR run. It has four *implicit coefficients*: $f(x) = w_0 + w_1*\log(w_2 + w_3*x)$ where $w_0 = 0.0$, $w_1 = 1.0$, $w_2 = 0.0$, and $w_3 = 1.0$. The first two coefficients w_0 and w_1 are linear; the others are nonlinear. As Figure 1 illustrates, GP should be able to make small steps in the space of the function's behavior by having all relevant coefficients readily available. If there is a coefficient shortage, tunability of the function is compromised.

Coefficient Overabundance. Missing constants in some places is one issue, and having too many in other places is another. The GP system is evolving more parameters than it needs to. Figure 2 illustrates one of many examples.

Non-compact Polynomials and Rationals. In GP, it takes many terms to build up a polynomial, and sometimes those terms cancel each other out causing redundant terms, as Figure 3 shows. In the literature, this is also handled implicitly as part of symbolic simplification.

Dimensional Awareness. In real-world use, functions describe *something*, and that "something" has units of measurement. Each input variable, and the target variable, has its own unit, such as "m/s" for a velocity variable. For a GP-evolved function to be physically meaningful, the units have to align, e.g. only like units can add, and the input variables must propagate through to the output such that the correct output unit is hit. Most SR systems ignore this, but the work of Keijzer is a notable exception: he demonstrated one system that used dimensionless values, another that biased evolution towards correct units, and a third system that had correct-by-construction units [9][10]. He did note that if there is a coefficient in front of an expression, that coefficient could conceivably have "corrective" units such that the input units translated properly into the output units. Interestingly, the existence of coefficients everywhere (implicit or explicit) causes implicit corrective unit transformations!

Bounded Ranges for Expression Outputs. For a given problem, each unit of measurement has a range of reasonableness. For example, velocity of a car can safely be bounded between 0 and 500 km/h. An ideal function would never allow intermediate or final expressions that go beyond unreasonable unit ranges. Most GP research ignores this, though Keijzer handles this via interval arithmetic in GP [10][11].

Bounded Ranges for Operators. Some mathematical operators are only valid for specific ranges, e.g. '/' can only have a nonzero denominator, and log() needs a positive argument. GP research typically handles this by "protected operators" [14] or simple exception handling, though the safest and most elegant way is probably interval arithmetic [10][11].

3. CAFFEINE

3.1 Canonical Form Functions

In this section we describe how we address each SR issue. Canonical form functions will play a key role.

We follow the tenets of (a) ensuring maximum expressiveness per node, and (b) making all individuals directly interpretable, i.e. not needing to manipulate expressions to simplify.

Figure 4 shows the general structure of a CAFFEINE canonical-form basis function. It has levels of expressions that alternate between linear and nonlinear, "gated" by nonlinear functions. The linear expressions are a sum of basis functions; each basis function gets a weight, plus there is one overall offset. A basis function is a combination of a polynomial/rational and/or one or more nonlinear operators. Inside each nonlinear operator is the next level of linear expressions. CAFFEINE places coefficients only where they are needed, and nowhere else. This emerges as a *canonical form* for functions.

An example is: $f(x) = -10.3 + 3.1*x_6 + 1.87 * x_1 * \log(-1.95 + 10.3 * (x_2*x_7)/(x_5))$. The value '-10.3' is the top linear offset coefficient w_o; the '3.1' and the '1.87' are the coefficients of basis functions for the top 'weighted linear add'; the 'x_6' is the lone instance of top-level standalone 'Poly/Ratl'; the 'x_1' is a 'Poly/Ratl' that has a product with the nonlinear function log(). Inside the log() is another weighted linear add subfunction.

Each 'Poly/Ratl' can be described in terms of the exponent for each input variable; therefore we achieve always-compact polynomials and rationals.

Typical usage of CAFFEINE would restrict the number of nonlinear operator layers to just one or two, which is a more effective "maximum depth" constraint, therefore resolving the issue of excessive compounding of nonlinear components. There can also be a limit on the maximum number of linear basis functions, easy to set because usually beyond 10 or so would be too much for humans to interpret, and fewer is fine. Thanks to its canonical form, all evolved functions are immediately interpretable, with no symbolic manipulation needed.

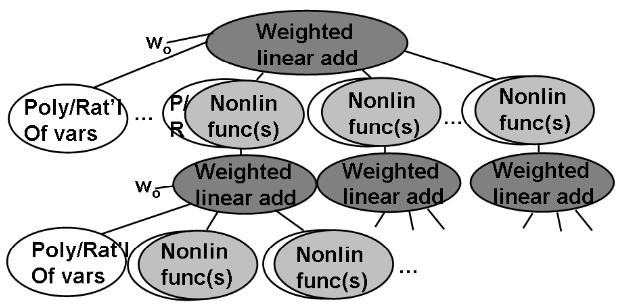

Figure 4: Caffeine evolves functions of this canonical form.

Such constraints on CAFFEINE directly resolve excessive complexity including bloat. Furthermore, they can be used in a complementary fashion with other complexity-reducing tactics, e.g. having a second objective of complexity.

The "only as needed" coefficient placement resolves coefficient overabundance and shortage issues. A coefficient on everything also means that they can be "dimensional transforms" to resolve dimensional awareness.

In determining coefficient values, we distinguish between linear and nonlinear coefficients. A CAFFEINE individual is a set of basis functions which are linearly added. Each basis function is a tree of grammatical derivations. Linear coefficients are found by evaluating each tree across all input samples to get a matrix of basis function outputs, then to apply least-squares regression with that matrix and the target output vector to find the optimal linear weights.

With each nonlinear coefficient in the genotype (i.e. ones that are not found via linear regression), a real value will accompany it, taking a value in the range $\left[-2*B, +2*B\right]$. During interpretation of the genome the value is transformed into $\left[-1e+B, -1e-B\right] \cup \left[0.0\right] \cup \left[1e-B, 1e+B\right]$. As for search for the value, any SR strategy might work. In this paper, we use mutation on a distribution, in particular a Cauchy distribution on all values at once which cleanly combines aggressive local tuning with the occasional large change [33].

To keep the approach simple, at this point we have not added the ideal method (interval analysis) for bounding ranges for expressions or operators. To handle bounded operators, we merely have exception handling on expression evaluation.

3.2 CAFFEINE Implementation via a Grammar

Here, we show how CAFFEINE can be implemented via an context-free grammar (CFG), but extended slightly to specially handle coefficients and Poly/Rationals.

Many SR systems have used CFGs [23], typically in a form like the following (though usually with fewer nonlinear operators). `EXPR` is the start symbol; possible expansions follow the `=>` symbol and are separated by |'s. The `W` is a coefficient, and can be implemented in one of many ways [5].

```
EXPR    => VAR | OP_1ARG ( EXPR ) |
           OP_2ARG ( EXPR , EXPR )
OP_2ARG => + | - | * | / | POW | MAX | ...
OP_1ARG => INV | LOG10 | SIN | ...
VAR     => X1 | X2 | ... | Xn | W
```

Importantly, there is no distinction in how the operators +, -, *, and / are used in comparison to the other operators, and variables and constants get directly plugged into expressions, giving rise to the SR issues given previously. More advanced non-CFG grammars handle more constraints [10], but do not make those distinctions either.

The CAFFEINE grammar is explicitly designed to create separate layers of linear and nonlinear functions and place coefficients and variables carefully:

```
REPVC   => VC | REPVC * REPOP | REPOP
REPOP   => REPOP * REPOP | OP_1ARG ( W + REPADD )|
           OP_2ARG ( 2ARGS ) | ... 3OP, 4OP etc
2ARGS   => W + REPADD, MAYBEW | MAYBEW, W + REPADD
MAYBEW  => W | W + REPADD
REPADD  => W * REPVC | REPADD + REPADD
OP_2ARG => DIVIDE |POW | MAX | ...
OP_1ARG => INV | LOG10 | ...
```

The start symbol is REPVC, which expands into one basis function (remember that an individual has several root-level basis functions). Note the strong distinction among operators. The root is a product of variables (REPVC) and / or nonlinear functions (REPOP). Within each nonlinear function is REPADD, the weighted sum of next-level basis functions.

A VC is a "variable combo", intended to maintain a compact representation of polynomials/rationals. Its expansion could have been implemented directly within the grammar; though in our baseline system we store a vector holding an integer value per design variable as the variable's exponent. An example vector is [1,0,-2,1], which means $(x_1 * x_4)/(x_3)^2$. This approach guarantees compactness and allows for special operators on the vector. The operators we use are: one-point crossover, and randomly adding or subtracting to an exponent value.

3.3 A Baseline CAFFEINE SR System

The baseline employs the direct tree-style grammar-based GP system of Whigham [32] following the CAFFEINE grammar. So, evolutionary operators must respect the derivation rules of the grammar, i.e. only subtrees with the same root can be crossed over, and random generation of trees must follow the derivation rules. (Section 5 will compare to Grammatical Evolution [23].)

Our main objective is to minimize root mean-squared error (RMSE), but as there is still value in a complexity bias (it just doesn't have to do as much), we employ a multiobjective algorithm, NSGA-II [4], with a second objective of minimizing complexity. "Complexity" is somewhat arbitrary but we make it dependent on the number of basis functions, the number of nodes in each tree, and the exponents of VCs:

$$\text{complexity}(f) = \sum_{j=1}^{M_f} \left(w_b + \text{nnodes}(j) + \sum_{k=1}^{\text{nvc}(j)} \text{vccost}(vc_{k,j})\right) \quad (1)$$

where w_b is a constant to give a minimum cost to each basis function, nnodes(j) is the number of tree nodes of basis function j, nvc(j) is number of VCs of basis function j, and

$$\text{vccost}(\text{vc}) = w_{vc} \sum_{\dim=1}^{d} \text{abs}(\text{vc}(\dim)) \cdot$$

We apply the usual Whigham-style crossover and mutation operators for trees. In addition, the basis function operators include: creating a new individual by randomly choosing >0 basis function from each of 2 parents; deleting a random basis function; adding a randomly generated tree as a basis function; copying a subtree from one individual to make a new basis function for another. Coefficient and VC operators are as described in sections 3.1 and 3.2, respectively.

3.4 CAFFEINE Search Biases

The fact that the CAFFEINE search space is more structured than simpler-grammar SR means that we can get a better idea of where the biases might be. CAFFEINE functions can be viewed sum-of-products (-of-sums-of-products-of…) expressions. While CAFFEINE could support product-of-sums, it currently does not, which means that CAFFEINE will have difficulty with functions better modeled as products of sums. The constraints of CAFFEINE also remove redundancies within functional space, which could also impact its search ability. Also, at this point we have been liberal in our choice of operators, but some might have damaging biases; a future challenge would be to identify each operator's relative effectiveness. Finally, CAFFEINE's extra emphasis on coefficients also makes it more sensitive to how coefficients are handled. In particular, the parameter mutation rate, and the chosen range of possible coefficient values could have large sensitivities.

4. APPLICATION: KNOWLEDGE EXTRACTION IN ANALOG DESIGN

This section summarizes the first application of CAFFEINE: knowledge extraction for analog circuits [17].

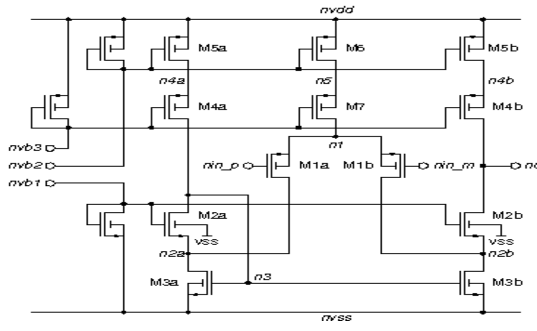

Figure 5: A CMOS Operational Transconductance Amplifier, used to generate datasets for six SR problems

A "symbolic model" of an analog circuit is merely an expression that maps circuit designables (like transistor widths and lengths, or circuit biases) to a circuit performance measure (like power consumption). Fundamentally, such models increase a designer's understanding of a circuit, which leads to better human decision-making in circuit sizing, layout, verification, and topology design. Since well-designed analog circuits are crucial to the multibillion-dollar semiconductor industry, automated approaches to symbolic model generation are of great interest. Over the last two decades, there has been active research on symbolic modeling. However, no previous tool could create models of arbitrary nonlinear analog circuits with accuracy in line with the "gold standard" of circuit analysis: SPICE circuit simulators.

Table 1: CAFFEINE-generated symbolic models of an analog circuit which gave <10% prediction error

Perf.	Symbolic Model
A_{LF}	-10.3 + 7.08e-5 / id1 +1.87 * ln(-1.95e+9 + 1.00e+10 / (vsg1*vsg3) +1.42e+9*(vds2*vsd5) / (vsg1*vgs2*vsg5*id2))
f_u	10^(5.68 - 0.03 * vsg1 / vds2 – 55.43 * id1+ 5.63e-6 / id1)
PM	90.5 + 190.6 * id1 / vsg1 + 22.2 * id2 / vds2
v_{offset}	- 2.00e-3
SR_p	2.36e+7 + 1.95e+4 * id2 / id1 - 104.69 / id2 + 2.15e+9 * id2 + 4.63e+8 * id1
SR_n	- 5.72e+7 - 2.50e+11 * (id1*id2) / vgs2 + 5.53e+6 * vds2 / vgs2 + 109.72 / id1

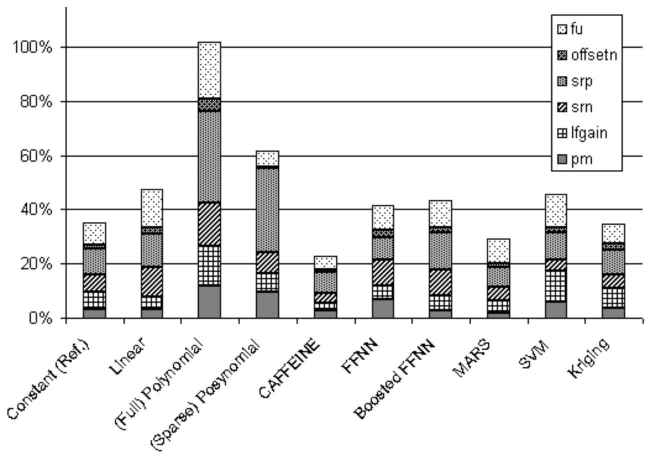

Figure 6: Across six analog circuit modeling problems, CAFFEINE-based GP has the best prediction compared to several state-of-the art approaches including neural networks [27], splines [7], and support vector machines [6]. The y-axis is prediction error summed across 6 problems.

In [17], the circuit of interest is as shown in Figure 5. The goal was to discover expressions for each of six performance measures: low-frequency gain (A_{LF}), unity-gain frequency (f_u), phase margin (PM), input-referred offset voltage (v_{offset}), and the positive and negative slew rate (SR_p, SR_n). These measures each had 13 possible input variables, which were the designables of the circuit (in this case, circuit biases). There were 243 training samples and 243 testing samples. Input samples were generated via Latin Hypercube Sampling [21], and corresponding outputs were found by circuit simulation to extract performance measures. No scaling was used.

The baseline CAFFEINE described in section 3.3 was applied on the datasets, to generate a set of models trading off error and complexity. Table 1 shows some of the models generated for each of these six symbolic modeling problems. Note how readily interpretable they are, of significance especially because they were not post-processed with any symbolic manipulation at all.

A good knowledge extractor *should* be able to predict well on unseen data, so [18] compared (the baseline) CAFFEINE's prediction abilities to other well-known techniques. As shown in Figure 6, CAFFEINE fared the best.

5. STUDY OF CAFFEINE VARIANTS

With promising results on the baseline, and convergence speed an active area in GP research, it is natural to ask how fast CAFFEINE variants might be. In this section, we test the effects of Grammatical Evolution (GE) [23], of grammar-defined introns, and of smooth, uniform crossover with smooth point mutation (SUX/SM) [26]. We also compare to simple-grammar Whigham-GP and simple-grammar GE.

5.1 CAFFEINE with Grammatical Evolution

Unlike Whigham-style GP which evolves trees, GE [23] uses bitstrings which sequentially specify derivation rules. We implemented GE using the standard operators of one-point crossover, flip-bit mutation, and codon duplication. We inserted minor modifications in order to have a more clean comparison to the baseline system. First, each GE individual had one bitstring for each basis function, and the linear weights were found via linear regression. Secondly, for the coefficients, rather than one of many possible GE-style approaches [5], we had a second list of numbers, where there is one real-valued number to correspond with each derivation rule. During evaluation, it follows a CAFFEINE-style log-expansion. The mutation operator for these values is also Cauchy mutation. Note that these real-valued numbers only get expressed when their corresponding `W` is being expressed. Third, VCs were implemented with the following derivation rules. Like the system in section 5.3, we constrained to a maximum two variable interactions, and biased to a variable exponent of 1.0.

```
VC     => POWVAR | POWVAR * POWVAR
POWVAR => VAR | VAR ^ POWEXP
VAR    => x1 | x2 | ... | xn
POWEXP => -2 | -1 | -0.5 | 0.5 | 2
```

5.2 CAFFEINE with Grammar-Defined Introns

In [23], the authors speculate that grammar-defined introns might be useful to GE, so we test that for context of a CAFFEINE system. CAFFEINE offers several opportunities to insert introns because of good control over coefficient placement. The grammar is modified as follows:

```
REPOP => REPOP * REPOP |
         OP_1ARG ( W + REPADD ) ^ OPEXP |
         OP_2ARG ( 2ARGS ) ^ OPEXP
OPEXP => 0 | 1
REPADD=> W_OR_ZERO * EXPR | REPADD + REPADD
W_OR_ZERO => W | 0.0
```

Thus, CAFFEINE allows introns in two ways: turning off nonlinear operators via power-of-zero, and turning off basis functions by forcing their weight to zero. Such grammar-defined introns can be part of Whigham-style or GE-style representation; we test both.

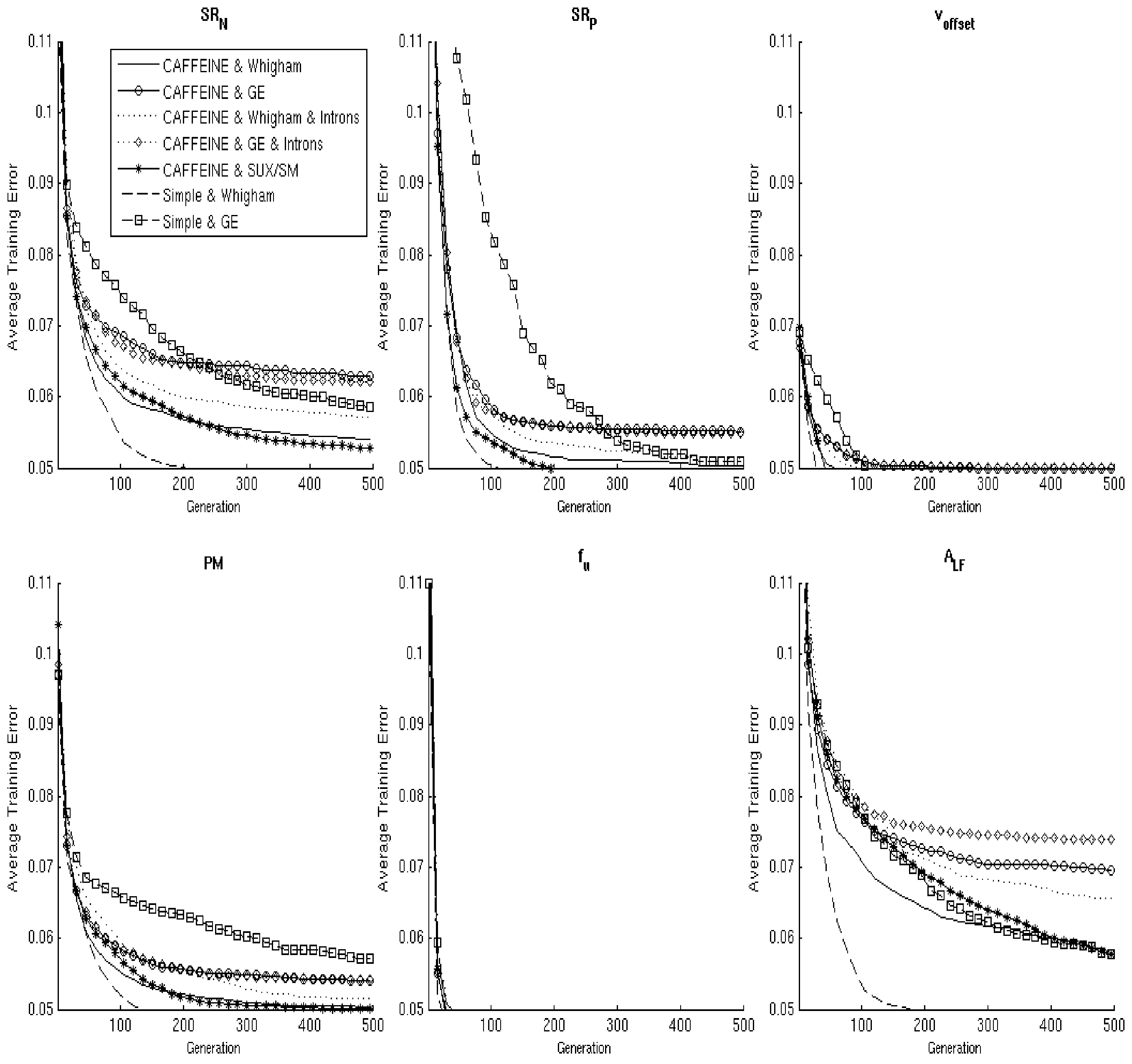

Figure 7: Each subplot shows the average (over 30 runs) of training RMSE vs. generation, for many symbolic regression variants. Each subplot is for a different circuit modeling problem.

5.3 CAFFEINE with Smooth, Uniform Crossover and Smooth Point Mutation

In [26], significant speedup in GP was achieved by leveraging SUX/SM for improved mixing of building blocks. That approach was applied to a logic synthesis problem, but it was not easy to apply to general GP problems, as one needs to align operators, handle different depths, and handle functions of different arity. Nevertheless, [19] showed that the structure of CAFFEINE lends itself well to uniform crossover. The trick is to pre-assign a fixed number of additions or multiplications for each level in the function structure, but allow them to be turned off by introns. This results in a fixed-size tree, allowing for ready application of SUX/SM operators. In [19] this variation performed very well compared to the baseline; that system is compared here too.

5.4 Experimental Setup

We test on the circuit application of section 4. Recall that there are six modeling problems; each problem has 13 input variables and 243 samples. For speed comparisons, it is simpler to just optimize on the single objective of training error, and compare that. Thus, all experiments in this section are for just single-objective EAs. The aim is to build models with <5% training error; at which point the EA run is considered successful and stopped. Operators allowed were: $\sqrt{x}$, $\log_{10}(x)$, $1/x$, abs(x), x^2, 10^x, max(x_1,x_2), min(x_1,x_2). Maximum number of basis functions = 7, population size 200, stop when 500 generations or target error hit, varcombo exponents in [-2,-1,-0.5,0,0.5,1,2], and weights in $[-1e+10,-1e-10] \cup [0] \cup [1e-10,1e+10]$. Maximum tree depth was 7, therefore allowing just one layer of nonlinear operators. For non-GE approaches, all operators had equal probability, except parameter mutation was 5x more likely. GE settings were in line with [23] as much as possible: each basis function genotype could have 1 to 10 integers, each in the range {0,2,…,255}; one-point crossover had a probability 90/135, parameter mutation 30/135, and the rest had equal probability among the remaining 15/135. Thirty runs were done for each approach on each problem.

5.5 Experimental Results

Remember that the benefit of CAFFEINE-style constraints is for the resulting function to be interpretable (such as those shown in section 4). To remind the reader of how ugly simple-GP-evolved expressions can be, here is a typical basis function found in the best individual of a simple-grammar A_{LF} run:

```
- 1.40 * ( vsg1 + max( vsg5, max( max( max( vsg5, max(
vsg3 + vgs2, min( vsg3, abs( 1/vds2 ) ) ) - log10(vsd5) ),
min( ib2, abs( sqrt( abs(id1) ) ) ) ) - log10(vsd5), max( id2,
min( vsg3, abs( sqrt( abs( log10(id2) ) ) ) ) ) + log10(vsd5) )
- min( vsg3, abs( sqrt( abs(id1) ) ) ) - log10(vsd5) ) )
```

In contrast, the CAFFEINE-style functions all had the same readily interpretable "look", thanks to the canonical forms that they were constrained to (see Table 1). This is the case no matter which CAFFEINE variant is used, whether it uses Whigham-style or GE, introns or not, or SUX/SM. So, the choice of using CAFFEINE can be largely orthogonal to other choices regarding the search algorithm.

We had two baseline non-CAFFEINE algorithms, one using a Whigham style of handling the grammar and one using GE style. Interestingly, these two baseline approaches were at opposite ends of the performance spectrum, with all CAFFEINE approaches in between.

Our primary question is to see if convergence can still be reasonable under the extra search space constraints imposed by CAFFEINE, compared to a simple grammar. Figure 7 shows convergence over time for each variant and each problem. To answer our primary question, we compare the best CAFFEINE variant, SUX/SM, with the best simple-grammar variant, Simple & Whigham. In 5 of 6 cases, CAFFEINE seemed to have the same rate of convergence, though with a time lag of about 20 – 50 generations. The remaining case was A_{LF}, which posed the most difficulty for all approaches, though less so for Simple & Whigham.

We are also interested in performance differences between Whigham-style and GE representations. There are three comparisons to make, each with a Whigham and a GE variant: CAFFEINE, CAFFEINE & introns, and simple grammar. In all three comparisons, GE does worse than its Whigham-style counterpart. This is most notable on the simple grammar; which we surmise is because that grammar needs more symbols than CAFFEINE to be as expressive, and GE is known to have issues with building block disruption. In fairness, we only used canonical GE of [23]; recent GE techniques aim to address disruption [24]. Also, how we handled coefficients with CAFFEINE & GE may have been detrimental.

As for the effect of grammar-defined introns: they never helped, tending to slow convergence by about 10%. The cost of explicit introns is "some extra coding segments" and the potential payoff is adding extra genotype diversity to escape local optima. At our population of 200, the cost / benefit was not positive, but it is possible that the balance may tip at smaller population sizes ([31] had benefit from introns at tiny population sizes). There are also potentially better ways to include introns into the search.

The SUX/SM CAFFEINE variant had a sizeable performance advantage over other CAFFEINE variants. This is not surprising, as it is grounded in GP theory to have improved mixing of building blocks. It is also important to note that the SUX/SM strategy is not available to simple-grammar SR; it was the particular structure of CAFFEINE that enabled it.

6. CONCLUSION

GP can evolve functions, making it ideally suited for knowledge extraction from datasets. Unfortunately, there is no technique in the literature that both (a) ensures that such evolved expressions are human-interpretable and (b) is simple to apply. This paper presented a technique called CAFFEINE that meets both goals, by constraining the search to a space of *canonical form* functions. These constraints can readily be embedded in a grammar-based GP system, or even implicitly.

We demonstrated CAFFEINE in the real-world knowledge extraction problem of analog circuit analysis, and also show that it out-predicts other state-of-the-art techniques like support vector machines.

We explored variants of CAFFEINE and compared them in terms of convergence ability for six test problems. We found that the fastest CAFFEINE variant is moderately slower than a simple (i.e. far less constrained) symbolic regression grammar which evolves very hard-to-interpret expressions. The fastest variant uses smooth, uniform crossover with smooth mutation. Thus, if raw speed is the only goal, we would not recommend CAFFEINE. But if interpretability of expressions is a concern, especially during evolution, then CAFFEINE would be useful.

We also found that Whigham-style variants did better than Grammatical-Evolution variants, and that grammar-defined introns did not help.

CAFFEINE can readily fit into machine-code GP [22], for an attractive combination of high speed and interpretable results. Other directions include using newer GE variants such as piGE [24], or combining CAFFEINE with ALP / interval arithmetic [10][11].

7. REFERENCES

[1] P.J. Angeline, Two Self-Adaptive Crossover Operators for Genetic Programming, *Advances in Genetic Programming 2*, MIT Press, Chapter 5, pp. 90-115.

[2] W. Banzhaf, Genotype-Phenotype Mapping and Neutral Variation – A Case Study in Genetic Programming, *PPSN III*, 1994, pp. 322-332.

[3] L. Breiman et al. *Classification and Regression Trees*. Chapman & Hall, New York, 1984.

[4] K. Deb et al, A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II, *Proc. PPSN VI*, Sept. 2000, pp. 849-858.

[5] I. Dempsey, M. O'Neill, and A. Brabazon, Meta-Grammar Constant Creation with Grammatical Evolution by Grammatical Evolution, *GECCO 2005*, 2005.

[6] H. Drucker et al., Support vector regression machines, *Adv. in Neural Information Proc. Sys. 9*, 1997, pp. 155-161.

[7] J.H. Friedman, Multivariate Adaptive Regression Splines, *Annals of Statistics 19*, March 1991, pp. 1-141.

[8] C. G. Johnson, Artificial Immune System Programming for Symbolic Regression, *EuroGP 2003*, Essex, UK, 2003.

[9] M. Keijzer, V. Babovic, Dimensionally Aware Genetic Programming, *GECCO 1999*, vol. 2, 1999.

[10] M. Keijzer. *Scientific Discovery using Genetic Programming*. PhD Thesis, Water & Environment and the Dept. for Mathematical Modelling, T.U. Denmark, 2001.

[11] M. Keijzer, Improving Symbolic Regression with Interval Arithmetic and Linear Scaling, *EuroGP 2003*, LNCS 2610, 2003, pp. 71-83.

[12] M. Keijzer, Scaled Symbolic Regression, *Genetic Programming and Evolvable Machines*, 5(3), Sept. 2004, pp. 259-269.

[13] E. Kirshenbaum, H.J. Suermondt, Using Genetic Programming to Obtain a Closed-Form Approximation to a Recursive Function, *GECCO 2004*, 2004, pp. 543-556.

[14] J.R. Koza. *Genetic Programmin*g. MIT Press, 1992.

[15] W.B. Langdon, R. Poli. *Foundations of Genetic Programming*. Springer, 2002.

[16] P. Langley et al. *Scientific Discovery, Computational Explorations of the Creative Process.* The MIT Press, 1987.

[17] T. McConaghy, T. Eeckelaert, and G. Gielen, CAFFEINE: Template-free Symbolic Model Generation of Analog Circuits via Canonical Form Functions and Genetic Programming, *Design Automation and Test Europe*, 2005.

[18] T. McConaghy, G. Gielen, Analysis of Simulation-Driven Numerical Performance Modeling Techniques for Application to Analog Circuit Optimization, *Proc. International Symposium on Circuits and Systems*, 2005.

[19] T. McConaghy, G. Gielen, Double-Strength CAFFEINE: Fast Template-Free Symbolic Modeling of Analog Circuits via Implicit Canonical Form functions and Explicit Introns, *Design Automation and Test Europe,* Mar 2006 (in press).

[20] B. McKay et al., Non-linear Continuum Regression Using Genetic Programming, in *GECCO 1999* (2), pp. 1106-1111.

[21] M.D. McKay, R.J. Beckman, and W.J. Conover, A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output From a Computer Code, *Technometrics*, 21(2), 1979, pp. 239-245.

[22] P. Nordin, A Compiling Genetic Programming System that Directly Manipulates the Machine Code, *Advances in Genetic Programming I*, MIT Press, 1994, pp.311-331.

[23] M. O'Neill, C. Ryan. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language.* Kluwer, 2003.

[24] M. O'Neill et al, Pi Grammatical Evolution, *GECCO 2004,* LNCS 3103, 2004, pp. 617-629.

[25] Panait, L., Luke, S., Alternative Bloat Control Methods, *GECCO 2004*, LNCS 3103, 2004, pp. 630-641.

[26] R. Poli, J. Page, W. B. Langdon, Smooth Uniform Crossover, Sub-Machine Code GP and Demes, *GECCO 1999*, vol 2, July 1999, pp. 1162-1169.

[27] D.E. Rumelhart et al., Learning Internal Representations by Error Propagation. *Parallel Distributed Processing Vol I*, MIT Press, 1986.

[28] G. Spencer, Automatic Generation of Programs for Crawling and Walking, *Advances in Genetic Programming*, MIT Press, Chapter 15, pp. 335-353.

[29] A.B. Tickle et. al, Lessons from Past, Current Issues and Future Research Directions in Extracting the Knowledge Embedded in Artificial Neural Networks, *Neural Hybrid Systems,* Springer Verlag, 1999.

[30] A. Topchy, W.F. Punch, Faster Genetic Programming Based on Local Gradient Search of Numeric Leaf Values, *GECCO 2001*, 2001, pp. 155-162.

[31] V. Vassilev, J. Miller, The Advantages of Landscape Neutrality in Digital Circuit Evolution, *ICES 2000*, pp. 252-263.

[32] P. A. Whigham, Grammatically-based Genetic Programming, *Proc. Workshop on GP: From Theory to Real-World Applications*, Tahoe City, CA, 1995.

[33] X. Yao, Y. Liu, G. Lin, Evolutionary Programming Made Faster, *IEEE Trans. Ev. Comp.* 3(2), July 1999, pp. 82-102.