Toward a Better Sine Wave

Matthew Streeter
Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609

Lee A. Becker
Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609

Abstract

In a previous paper, we showed that genetic programming can be used to evolve approximations to functions which, given certain trade-offs between cost and error, are superior to Padé approximations, which represent a generalization of Taylor series and a powerful technique from numerical analysis. In the present paper, we present an extension to this work which allows existing Padé approximations to be used to bootstrap the evolutionary process. Specifically, we use program trees corresponding to existing Padé approximations to seed the initial GP population; this allows for the evolution of approximations which are much more accurate in approximating certain combinations of function and interval. We illustrate the effectiveness of this approach by evolving approximations to the function $\sin(x)$ over the interval $[0,\pi/2]$ which build upon and significantly improve some of the most efficient Padé approximations to that function.

1 INTRODUCTION

In a previous paper (Streeter and Becker 2001), we presented a number of results concerning the application of genetic programming to the discovery of numerical approximation formulae. Firstly, we evolved variations on the first three terms of the asymptotic expansion for the Harmonic number series. Secondly, we were able to evolve rational polynomial approximations which, given certain trade-offs between cost and error, were superior to Padé approximations, and to evolve approximations to certain functions of more than one variable to which the Padé approximation technique cannot be applied. Finally, we demonstrated that evolved approximations can be refined through the evolution of approximations to their error function. In the present paper, we provide a novel extension to our previous work which allows existing approximations to be used to jump-start the evolutionary process. Specifically, we take program trees corresponding to existing approximating expressions and use them as seed individuals in the initial GP population. We illustrate this technique by evolving rational polynomial approximations to the function sin(x) over the interval $[0,\pi/2]$. As will be shown, this extension allows for the evolution of approximations which are orders of magnitude more accurate than those which would be obtained through a straightforward application of GP, and allows us to evolve a number of approximations which, for given levels of cost and error, are superior to Padé approximations, despite the fact that Padé approximations are highly suited to this combination of function and interval.

This paper consists of an introduction, copied directly from our previous paper, followed by a description of our general experimental technique, a discussion of our technique for building upon existsing approximations, and a description of our experiments involving evolution of sine wave approximations.

1.1 MOTIVATIONS

Numerical approximation formulae are useful in two primary areas: firstly, approximation formulae are used in industrial applications in a wide variety of domains to reduce the amount of time required to compute a function to a certain degree of accuracy (Burden and Faires 1997), and secondly, approximations are used to facilitate the simplification and transformation of expressions in formal mathematics. The discovery of approximations used for the latter purpose generally requires human intuition and insight, while approximations used for the former purpose tend to be polynomials or rational polynomials obtained by a technique from numerical analysis such as Padé approximants (Baker 1975; Bender and Orszag 1978) or Taylor series. Genetic programming (Koza 1992) provides a unified approach to the discovery of approximation formulae which, in addition to having the obvious benefit of automation, provides a power and flexibility that potentially allows for the evolution of approximations superior to those obtained using existing techniques from numerical analysis.

1.2 EVALUATING APPROXIMATIONS

In formal mathematics, the utility or value of a particular approximation formula is difficult to analytically define, and depends perhaps on its syntactic simplicity, as well as the commonality or importance of the function it approximates. In industrial applications, in contrast, the value of an approximation is uniquely a function of the

computational cost involved in calculating the approximation and the approximation's associated error. In the context of a specific domain, one can imagine a utility function which assigns value to an approximation based on its error and cost. We define a reasonable utility function to be one which always assigns lower (better) scores to an approximation a_1 which is *unequivocally superior* to an approximation a_2, where a_1 is defined to be unequivocally superior to a_2 iff. neither its cost nor error is greater than that of a_2, and at least one of these two quantities is lower than the corresponding quantity of a_2. Given a set of approximations for a given function (obtained through any number of approximation techniques), one is potentially interested in any approximation which is not *unequivocally inferior* (defined in the natural way) to any other approximation in the set. In the terminology of multi-objective optimization, this subset is referred to as a Pareto front (Goldberg 1989). Thus, the Pareto front contains the set of approximations which could be considered to be the most valuable under some reasonable utility function.

1.3 RELATED WORK

The problem of function approximation is closely related to the problem of function identification or symbolic regression, which has been extensively studied by numerous sources including (Koza 1992; Andre and Koza 1996; Luke and Spector 1997; Nordin 1997; Ryan, Collins, and O'Neill 1998). Approximation of specific functions has been performed by Keane, Koza, and Rice (1993), who use genetic programming to find an approximation to the impulse response function for a linear time-invariant system, and by Blickle and Thiele (1995), who derive three analytic approximation formulae for functions concerning performance of various selection schemes in genetic programming. Regarding general techniques for the approximation of arbitrary functions, Moustafa, De Jong, and Wegman (1999) use a genetic algorithm to evolve locations of mesh points for Lagrange interpolating polynomials.

2 EVOLVING NUMERICAL APPROXIMATION FORMULAE USING GENETIC PROGRAMMING

The experiments reported in this paper make use of the standard genetic programming paradigm as described by Koza (1992). Our task is to take a function in symbolic form (presented to the system as a set of training points) and return a (possibly singleton) set of expressions in symbolic form which approximate the function to various degrees of accuracy. In the experiments reported in this paper, this task is treated as a symbolic regression problem in which the function set is limited in such a way that the search space contains only approximations, rather than exact matches, to the target function.

The system used for the experiments described in this paper was designed to be functionally equivalent to that described by Koza (1992) with two minor modifications. Firstly, the evolution of approximation formulae requires the cost of each approximation to be computed. We accomplish this by assigning a raw cost to each function in the function set, and taking the cost of an approximation to be the sum of the functional costs for each function node in its expression tree whose set of descendent nodes contains at least one input variable. For all experiments reported in this paper, the function costs were somewhat arbitrarily set to 1 for the multiplication and division functions, and 0.1 for addition and subtraction.

Secondly, rather than simply reporting the best (i.e. most accurate) approximation evolved in each of a number of runs, we report the Pareto front for the union of the population histories of each independent run, computed iteratively and updated at every generation. Thus, this system returns the subset of approximations which are potentially best (under some reasonable utility function) from the set of all approximations evolved in the course of all independent runs.

3 BUILDING UPON EXISTING APPROXIMATIONS

A straightforward way to make use of known approximations in evolving new ones is to create program trees corresponding to known approximations and insert them as individuals in the initial GP population. In our system, the insertion of seed individuals is built into the random tree generation algorithm: when initializing the root node, we add one additional "terminal" for each seed individual to the set of available functions and terminals. If this "terminal" is chosen, the seed individual becomes the entire program tree. This is equivalent to inserting each of a set of s seed individuals with a frequency $1/(s+r)$, where r is the number of other available elements — in our case the four arithmetic operators, the variable X, and the random numeric terminal. The only restriction imposed here on the seed individuals is that they be defined using the same primitive operations that are present in the function set for evolving approximations. As a practical matter, it is also necessary to set any size and depth limits in such a way that the seed individuals are not immediately killed off.

4 EVOLVING SINE WAVE APPROXIMATIONS

4.1 THE PARETO FRONT FOR PADÉ APPROXIMATIONS

In comparing evolved approximations to Padé approximations or in building upon Padé approximations in the course of the evolutionary process, we are interested in the set of Padé approximations which are potentially the most valuable, i.e. the Pareto front for the set of all Padé approximations. Since the number of Padé

approximations to a given function is infinite, we have restricted our attention to Padé approximations of numerator and denominator degree 20 or less. Additionally, we have made the reasonable choice of taking x=0 as the point about which the Padé approximations are centered. For each of the $21^2 = 441$ approximations under consideration, we have calculated values for cost and error using the Maple symbolic mathematics package. In calculating the error, we use a Riemann integral with 100 points. In calculating cost, we first put the expression into continued-fraction form (known to minimize the number of necessary multiplications and divisions), then evaluate the minimum number of multiplications/divisions required to compute the expression using a custom Maple procedure. This custom procedure operates by first counting the number of multiplications and divisions nominally present in the expression, then subtracting for redundant computations of powers of the variable x, e.g. x^2+x^3 nominally requires three multiplications, but can actually be computed using only two, since the computed value of x^2 can be used to compute x^3. Note that this cost metric differs from that actually used during the evolution in that it attempts to determine the *minimum* number of multiplications and divisions that are necessary, and also that it assigns no cost to additions or subtractions. Using this method, and taking the Pareto front with respect to the calculated cost and error values, we obtain the 20 Padé approximations given in Table 1. In this and all other tables in this paper, numeric values (including both constants occurring in expressions and error values) have been reported to 8 significant digits of accuracy.

4.2 EVOLVED APPROXIMATIONS

In our experiments in evolving approximations to sin(x) over the interval [0,π/2], we made use of all 20 Padé approximations given in Table 1, taken one at a time as seed individuals for 20 sets of 50 independent GP runs. In each case, the approximation under consideration comprised an expected fraction 1/(1+6) ≈ 14.3% of the initial population. Parameter settings for all runs were the same as those presented as the defaults by Koza (1992) in his first genetic programming book, including a population size of 500, with two exceptions. First, tournament selection was used in place of fitness-proportionate selection, with a tournament size of 3. Second, in cases where the depth of the seed individual for a run exceeded 17 (the normal depth limit), the depth limit was set to the seed individual's depth plus 2 (to allow some possibility for growth). Training data for all experiments was taken as 100 function points uniformly spaced over the interval [0,π/2], and fitness was taken as the sum of absolute error over these training points.

Table 1: Padé Approximations for sin(x).

PADÉ APPROXIMATION			
NUM. DEGREE	**DENOM. DEGREE**	**COST**	**ERROR**
1. (-.19742607e-19*x^19+.44328679e-16*x^17-.42340571e-13*x^15+.22096123e-10*x^13-.67748181e-8*x^11+ .12286109e-5*x^9-.12680091e-3*x^7+.67979580e-2*x^5-.15718170*x^3+x)/(1+.94849669e-2*x^2+.45452521e-4*x^4 +.14582067e-6*x^6+.99417208e-15*x^12+.12036244e-17*x^14+.11293006e-20*x^16+.75114010e-24*x^18+ .65755338e-12*x^10+.34924931e-9*x^8+.27305462e-27*x^20)			
20	20	20	.22915571e-43
2. (-.23483895e-19*x^19+.50385990e-16*x^17-.46414075e-13*x^15+.23542477e-10*x^13-.70621153e-8*x^11+ .12600134e-5*x^9-.12854445e-3*x^7+.68390502e-2*x^5-.15744502*x^3+x)/(.42657643e-4*x^4+.1308174e-6*x^6+1 +.92216446e-2*x^2+.29536702e-9*x^8+.51362484e-12*x^10+.71261529e-18*x^14+.50968042e-21*x^16+ .19660866e-24*x^18+.69467729e-15*x^12)			
19	18	19	.87514396e-41
3. (.14208018e-16*x^17-.23003413e-13*x^15+.15469651e-10*x^13-.54927137e-8*x^11+.10910479e-5*x^9- .11925501e-3*x^7+.66213262e-2*x^5-.15605276*x^3+x)/(.10613904e-1*x^2+.11373697e-11*x^10+.54658702e-9 *x^8+.56976828e-4*x^4+.20462487e-6*x^6+.23530870e-17*x^14+.2105904e-20*x^16+.10369598e-23*x^18+ .18646697e-14*x^12+1)			
17	18	18	.14248440e-37
4. (.16867471e-16*x^17-.25985079e-13*x^15+.16814871e-10*x^13-.57983636e-8*x^11+.11272336e-5*x^9- .1213606e-3*x^7+.66722038e-2*x^5-.15638181*x^3+x)/(.53012726e-4*x^4+1+.1804405e-6*x^6+.10284854e- 1*x^2+.4480091e-9*x^8+.11782802e-14*x^12+.11464987e-17*x^14+.60358806e-21*x^16+.84035516e-12*x^10)			
17	16	17	.44316053e-35
5. (-.80822061e-14*x^15+.90614326e-11*x^13-.40881899e-8*x^11+.92888533e-6*x^9-.10997019e- 3*x^7+.63989808e-2*x^5-.15461957*x^3+x)/(1+.120471e-1*x^2+.73497474e-4*x^4+.29958315e-6*x^6+.90264195e- 9*x^8+.20859715e-11*x^10+.36668815e-14*x^12+.45656122e-17*x^14+.31513721e-20*x^16)			

15	16	16	.5799448e-32
6. (-.9571573e-14*x^15+.1015912e-10*x^13-.43997467e-8*x^11+.97045091e-6*x^9-.11255425e-3*x^7+.64635711e-2*x^5-.15504244*x^3+x)/(1+.11624224e-1*x^2+.6760839e-4*x^4+.25798483e-6*x^6+.70683436e-9*x^8+.14121755e-11*x^10+.19402017e-14*x^12+.14457592e-17*x^14)			
16	14	15	.14349330e-29
7. (.35219914e-11*x^13-.25957644e-8*x^11+.73675629e-6*x^9-.98292534e-4*x^7+.61106609e-2*x^5-.15274041*x^3+x)/(1+.13926254e-1*x^2+.98369951e-4*x^4+.46303746e-6*x^6+.15933641e-8*x^8+.40831791e-11*x^10+.7377696e-14*x^12+.74522308e-17*x^14)			
13	14	14	.14690078e-26
8. (.41578831e-11*x^13-.28831629e-8*x^11+.78364837e-6*x^9-.10151674e-3*x^7+.61952755e-2*x^5-.15330387*x^3+x)/(1+.13362795e-1*x^2+.89074689e-4*x^4+.38510777e-6*x^6+.11602817e-8*x^8+.23620428e-11*x^10+.26092061e-14*x^12)			
13	12	13	.28073332e-24
9. (-.11285502e-8*x^11+.5100612e-6*x^9-.83221836e-4*x^7+.57221746e-2*x^5-.15016988*x^3+x)/(1+.16496788e-1*x^2+.13830592e-3*x^4+.76861415e-6*x^6+.30545907e-8*x^8+.84628888e-11*x^10+.13225865e-13*x^12)			
11	12	12	.2171149-21
10. (-.13267047e-8*x^11+.55986494e-6*x^9-.87296681e-4*x^7+.5837572e-2*x^5-.15095788*x^3+x)/(1+.15708787e-1*x^2+.1223699e-3*x^4+.6044395e-6*x^6+.19466688e-8*x^8+.33875822e-11*x^10)			
11	10	11	.30842974e-19
11. (.25146483e-6*x^9-.63202655e-4*x^7+.51711181e-2*x^5-.14644332*x^3+x)/(1+.20223351e-1*x^2+.20834323e-3*x^4+.14059908e-5*x^6+.64407374e-8*x^8+.16784074e-10*x^10)			
9	10	10	.17196709e-16
12. (.29387219e-6*x^9-.68312274e-4*x^7+.53369556e-2*x^5-.14762311*x^3+x)/(1+.19043554e-1*x^2+.17754789e-3*x^4+.99546023e-6*x^6+.29673945e-8*x^8)			
9	8	9	.17246431e-14
13. (-.3594461e-4*x^7+.4330929e-2*x^5-.14056444*x^3+x)/(1+.26102228e-1*x^2+.34796697e-3*x^4+.29440163e-5*x^6+.14226233e-7*x^8)			
7	8	8	.6488747e-12
14. (-.41632773e-4*x^7+.45855159e-2*x^5-.14252345*x^3+x)/(1+.24143212e-1*x^2+.27605127e-3*x^4+.15950359e-5*x^6)			
7	6	7	.4269605e-10
15. (.29035821e-2*x^5-.12995655*x^3+x)/(1+.36710114e-1*x^2+.68860107e-3*x^4+.72619288e-5*x^6)			
6	6	6	.97933708e-8
16. (.33128908e-2*x^5-.13383838*x^3+x)/(1+.32828283e-1*x^2+.45093795e-3*x^4)			
5	4	5	.37808176e-6
17. (-.10544218*x^3+x)/(1+.6122449e-1*x^2+.18707483e-2*x^4)			
3	4	4	.44176819e-4
18. (-.11666667*x^3+x)/(1+.5e-1*x^2)			
3	2	3	.83028076e-3
19. x/(1+.16666667*x^2)			
1	3	2	.032510528
20. x			
1	0	0	.22923809

Table 2: Evolved Approximations for sin(x).

EVOLVED APPROXIMATION	SEED INDV.	COST	ERROR
1. (.16867471e-16*x^17-.25985079e-13*(.18044050e-6*x^3+x^12+.44800910e-9*x^20+.13508967e-32+.84035631e-12*x^5+.60358806e-21*x^4)*x^3+.16814871e-10*x^13-.57983636e-8*x^11+.11272336e-5*x^9-.12136060e-3*x^7+.66722038e-2*x^5-.15638181*x^3+x)/(.53012726e-4*x^4+1+.18044050e-6*x^6+.10284854e-1*x^2+.44800910e-9*x^8+.11782802e-14*x^12+.11464987e-17*x^14+.60358806e-21*x^15+.84035516e-12*x^10)	(*4*)	30	.11599524e-19
2. (.16867471e-16*x^17-.25985079e-13*x^15+.16814871e-10*x^13-.57983636e-8*x^11+.11272336e-5*x^9-.12136060e-3*x^7+.66722038e-2*x^5-.15638181*x^3+x)/(.53012726e-4*x^4+1+.18044050e-6*x^6+.10284854e-1*x^2+.44800910e-9*x^8+.11782802e-14*x^12+.11464987e-17*x^14+.21525401e-20*x^13+.84035516e-12*x^10)	(*4*)	19	.13399751e-19
3. (.16867471e-16*x^17-.25985079e-13*(.18044050e-6*x^5+x^12+.84035516e-12*x^10)*x^3+.16814871e-10*x^13-.57983636e-8*x^11+.11272336e-5*x^9-.12136060e-3*x^7+.66722038e-2*x^5-.15638181*x^3+x)/(.53012726e-4*x^4+1+.18044050e-6*x^6+.10284854e-1*x^2+.44800910e-9*x^8+.11782802e-14*x^12+.11464987e-17*x^14+.60358806e-21*x^15+.84035516e-12*x^10)	(*4*)	18	.15835320e-19
4. (.35219914e-11*x^13-.25957644e-8*x^11+.73675629e-6*x^9-.98292534e-4*x^7+.61106609e-2*x^5-.15274041*x^3+x)/(1+.13926254e-1*x^2+.98369951e-4*x^4+.46303746e-6*x^6+.15933641e-8*x^8+.40831791e-11*x^10+.73776960e-14*(-.25957644e-8*(.73675629e-6*x^8+2.0061107*x^6+x)*x-.22506492e-6*x^3+.45020678e-8*x^5+.73675629e-6*x+.73675629e-6*x^6)*x-.53427535e-24*x^7+.54355639e-20*x+.73776960e-14*x^12+.74522308e-17*x^14)	(*7*)	14	.19957714e-19
5. (.41578831e-11*(.41578831e-11*x^2-.28831652e-8*x^5+x^11)*x^2-.28831629e-8*x^11+.78364837e-6*x^9-.10151674e-3*x^7+.61952755e-2*x^5-.15330387*x^3+x)/(1+.13362795e-1*x^2+.89074689e-4*x^4+.38510777e-6*x^6+.11602817e-8*x^8+.23620428e-11*x^10+.26092061e-14*x^12)	(*8*)	13	.34188094e-19
6. (-.13267047e-8*x^11+.55986494e-6*x^9-.87296681e-4*x^7+.58375720e-2*x^5-.15095788*x^3+x)/(1+.15708787e-1*x^2+.12236990e-3*x^4+.60443950e-6*x^6+.19466688e-8*x^8+.33875822e-11*x^10)	(*10*)	11	.46211450e-19
7. (.25146483e-6*x^9-.63202655e-4*x^7+.51711181e-2*x^5-.14644332*x^3+x)/(1+.20223351e-1*x^2+.20834323e-3*x^4+.14059908e-5*x^6+.64407374e-8*x^8+.16784074e-10*(.25146483e-6*x^5+x^8)*x^2)	(*11*)	10	.11906981e-16
8. (.29387219e-6*x^9-.68312274e-4*x^7+.53369556e-2*x^5-.14762311*x^3+x)/(1+.19043554e-1*x^2+.17754789e-3*x^4+.99546023e-6*x^6+.29673945e-8*x^8+.15970628e-15*x^9)	(*12*)	9	.89135991e-15
9. (-.35944610e-4*x^7+.43309290e-2*x^5-.14056444*x^3+x)/(1+.026102228*x^2+.34796697e-3*x^4+.29440163e-5*(-.12507537e-7*x^4+x^3)*x^3+.14226233e-7*x^8)	(*13*)	8	.52074457e-12
10. (-.41632765e-4*x^7+.0045855159*x^5-.14252345*x^3+x)/(1+.024143212*x^2+.27605127e-3*x^4+.15950359e-5*x^6)	(*14*)	7	.22289908e-10
11. (.0029035821*x^5-.12995655*x^3+x)/(1+.036710114*x^2+.68860107e-3*x^4+.72619288e-5*x^5*(.68860107e-3+x))	(*15*)	6	.56504028e-8
12. (.0033128908*x^5-.13383838*x^3+x)/(1+.32828283e-1*x^2+.45093795e-3*x^2*(.45093795e-3*x^3-.45093862e-3+x^2))	(*16*)	5	.15708951e-6
13. (-.10544218*x^3+x)/(1+.061224490*x^2+.0018707483*(.061224490*x+.93877551*x^2)*x^2)	(*17*)	4	.95235444e-5
14. (-.11666667*x^3/(1+.005*x)+x)/(1+.05*x^2)	(*18*)	3	.29332169e-3
15. x/(1+.20800907*x^2)	(*19*)	2	.012285190

16. 3.8938800*x/(2*x+2.6285592)	(*7*)	1	.047321306
17. x	(*20*)	0	.22923809

For each experiment (i.e. each set of 50 runs), our system returned the Pareto front of the combined population histories of all 50 runs as output, which we then imported into a Maple worksheet and re-evaluated using the cost and error procedures described in the previous section. The Pareto front was then re-computed with respect to this new cost and error data. Finally, the Pareto front of the union of the Pareto fronts obtained from our 20 separate experiments was computed, resulting in the set of 17 approximations given in Table 2. This Pareto front, along with the Pareto front for Padé approximations, is presented graphically on the (cost,error) plane in Figure 1, where the dotted line connects points on the Pareto front for Padé approximations, and the solid line connects points on the Pareto front for evolved approximations. To make this figure readable, the error values are presented on a logarithmic scale.

4.3 COMPARISON OF PADÉ AND EVOLVED APPROXIMATIONS

A comparison of tables 1 and 2 reveals that 9 of the 17 approximations on the evolved approximations' Pareto front are more accurate than Padé approximations of the same cost, with one evolved approximation (approximation 13) representing a fourfold improvement in accuracy. Moreover, in all but 2 of these cases, the improvement arises not just from tweaking of constants but from substantial modification of the seed expression. The number of Padé approximations for which we can evolve improvements has been limited in part by our use of long double (80-bit floating point) arithmetic, which limits the accuracy of evolved approximations to an order of roughly 1.0e-19, as illustrated by the sharp peak in the graph of the solid line in Figure 1. For all evolved approximations whose error exceeds 1.0e-19, excepting the trivial approximation with cost of 0, we have evolved approximations which are substantially more accurate than Padé approximations of the same cost. The accuracy of these approximations, that of the corresponding Padé approximations, and the ratio of improvement are given in Table 3.

In a control experiment performed using no seed individuals, the most accurate approximation required 4 multiplications and had an error of 0.2222508. Thus, using Padé approximations as seed individuals has allowed us to increase this accuracy by many orders of magnitude.

We also conducted an experiment in which all Padé approximations in Table 1 were used as seed individuals at once, but found this tended to produce improvements to only the best seed individuals, rather than all 20.

Table 3: Comparison of Padé and Evolved Approximations Whose Error Exceeds 1.0e-19

COST	PADÉ ERROR (E_p)	EVOLVED ERROR (E_e)	E_p/E_e
10	.17196709e-16	.11906981e-16	1.4442543
9	.17246431e-14	.89135991e-15	1.9348448
8	.6488747e-12	.52074457e-12	1.2460491
7	.4269605e-10	.22289908e-10	1.9154879
6	.97933708e-8	.56504028e-8	1.7332164
5	.37808176e-6	.15708951e-6	2.4067919
4	.44176819e-4	.95235444e-5	4.6386951
3	.83028076e-3	.29332169e-3	2.8306149
2	.032510528	.012285190	2.6463187
1	NA	.047321306	NA
0	.22923809	.22923809	1.0

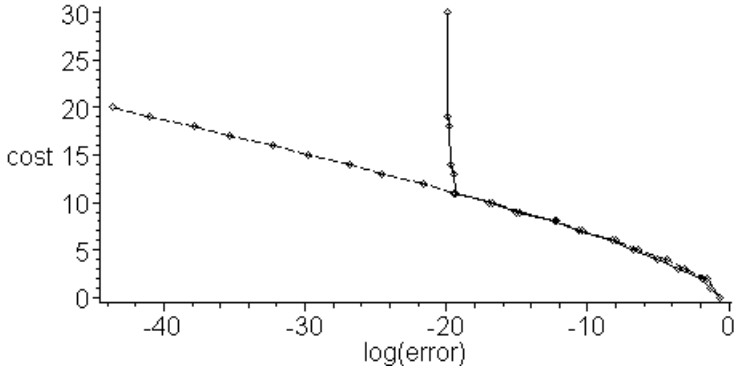

Figure 1: Pareto Fronts for Padé and Evolved Approximations.

5 FUTURE WORK

The work presented in this paper suggests a number of possible extensions. First, expressions other than Padé approximations could be used as seed individuals, e.g. minimax or Remez approximations. It would be interesting to see which types of approximations were more or less susceptible to refinement in this manner. Secondly, this technique could be applied to approximations other than rational polynomials; it effectively provides a general technique for attempting to refine any known approximation. Thirdly, we note that although what we desire when building upon an approximation is both to increase accuracy and to reduce cost, our fitness function scores individuals based only on accuracy. Thus, an appropriate alteration to the fitness function would presumably increase the likelihood of obtaining an improvement with respect to both objectives. Finally, these results could also presumably be improved by using more recent GP features such as automatically-defined functions (Koza 1994) and architecture-altering operations (Koza 1999).

6 CONCLUSIONS

In this paper, we have presented a general technique for evolving approximations which build upon existing, known approximations, with the possibility of improving both their accuracy and associated cost. We have illustrated this technique by evolving a number of improvements to Padé approximations of the function $\sin(x)$ over the interval $[0,\pi/2]$. Though it has not been explicitly shown in this paper, we expect that this approach would provide a general technique for improving the accuracy of existing approximations defined in terms of any set of primitive operations, in addition to rational polynomials. Finally, we have conjectured that with appropriate modifications to the fitness function, we could likely simultaneously improve both the cost and error for a wide variety of approximations presented as input to the system.

References

D. Andre and J. R. Koza (1996). Parallel genetic programming: A scalable implementation using the transputer network architecture. In P. J. Angeline and K. E. Kinnear, Jr. (eds.), *Advances in Genetic Programming 2*, 317-338. Cambridge, MA: MIT Press.

G. A Baker (1975). *Essentials of Padé Approximants*. New York: Academic Press.

C. M. Bender and S. A. Orszag (1978). *Advanced Mathematical Methods for Scientists and Engineers*. New York: McGraw-Hill.

T. Blickle and L. Thiele (1995). A comparison of selection schemes used in genetic algorithms. TIK-Report 11, TIK Institut fur Technische Informatik und Kommunikationsnetze, Computer Engineering and Networks Laboratory, ETH, Swiss Federal Institute of Technology.

R. L. Burden and J. D. Faires (1997). *Numerical Analysis*. Pacific Grove, CA: Brooks/Cole Publishing Company.

D. E. Goldberg (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley.

M. A. Keane, J. R. Koza, and J. P. Rice (1993). Finding an impulse response function using genetic programming. In *Proceedings of the 1993 American Control Conference*, 3:2345-2350.

J. R. Koza (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press.

J. R. Koza (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. Cambridge, MA: MIT Press.

J. R. Koza, D. Andre, F. H Bennett III, and M. A. Keane. (1999). *Genetic Programming 3: Darwinian Invention and Problem Solving*. San Mateo, CA: Morgan Kaufman.

S. Luke and L. Spector (1997). A comparison of crossover and mutation in genetic programming. In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. sIba, and R. L. Riolo (eds.), *Genetic Programming 1997: Proceedings of the Second Annual Conference*, 240-248. San Mateo, CA: Morgan Kaufmann.

R. E. Moustafa, K. A. De Jong, and E. J. Wegman (1999). Using genetic algorithms for adaptive function approximation and mesh generation. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith (eds.), *Proceedings of the Genetic and Evolutionary Computation Conference*, 1:798. San Mateo, CA: Morgan Kaufmann.

P. Nordin (1997). *Evolutionary Program Induction of Binary Machine Code and its Applications*. PhD thesis, der Universitat Dortmund am Fachereich Informatik.

C. Ryan, J. J. Collins, and M. O'Neill (1998). Grammatical evolution: Evolving programs for an arbitrary language. In W. Banzhaf, R. Poli, M. Schoenauer, and T. C. Fogarty (eds.), *Proceedings of the First European Workshop on Genetic Programming*, 1391:83-95. New York: Springer-Verlag.

M. Streeter and L. A. Becker (2001). Automated discovery of numerical approximation formulae via genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference*. San Mateo, CA: Morgan Kaufmann.