The Thirty-Second International Florida
Artificial Intelligence Research Society Conference (FLAIRS-32)

Synthesis of Limit Problems for Single-Variable Calculus

Blake Glueck
Bradley University
bglueck@mail.bradley.edu

Chris Alvin
Furman University
chris.alvin@furman.edu

Abstract

This paper presents a method for generating single-variable limit problems for an introductory Calculus course. Our method generates problems in two steps. The first step uses an evolutionary approach to construct unique functions f. The second step involves an analysis of f to compute distinct "approach" values. Our simulations demonstrate the limitations and utility of our approach.

1 Introduction

Calculus textbooks are replete with quality limit problems; however, no explicit techniques exist for generating fresh problems that do not adhere to common problem templates (Singh, Gulwani, and Rajamani 2012). Functions in limit problems tend to evidence a similar, somewhat simple structure; hence, textbook problems may begin to present little challenge for a fastidious student. We present a technique for generating fresh limit problems that deviate from standard textbook structure.

In this paper, we consider limit problems of the form $\lim_{x\to a} f(x)$ where f is both a function and expression defined by independent variable x. Our generation technique operates in two steps. We first generate a function f using an evolutionary technique where each function f is an individual in the population of all functions. Second, for f we compute 'approach' values (a in the limit expression).

Consider the expressions $A = \ln^2(x)/(53-x) - |x|$ and $B = \sqrt{1-x}/(x+3)$. Each expression can be represented as a syntax tree following standard notions of mathematical operator precedence and associativity we call an *expression tree*; see Figure 1 and Figure 2, respectively. We note exponential expressions (b^n) are expressed as a binary function $exp(b, n)$ in Figure 2 and generally roots are represented as rational exponents in our expression language.

Evolutionary Operations. Our technique operates with standard evolutionary operations (*selection*, *crossover*, and *mutation*) and evaluates the relative merit of individual functions using a *fitness function*.

Selection. We select the most fit individuals according to a fitness function fit. The larger the fitness score, the greater the probability that an individual will be a member of the next generation.

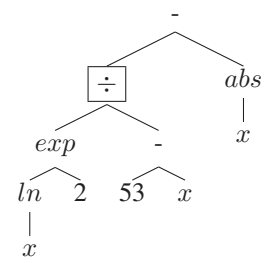

Figure 1: $A = \frac{\ln^2(x)}{(53-x)} - |x|$ as an Expression Tree

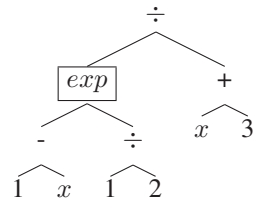

Figure 2: $B = \frac{(1-x)^{1/2}}{x+3}$ as an Expression Tree

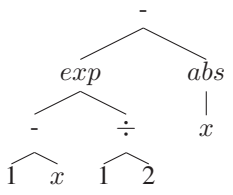

Figure 3: $\mathsf{crossover}(A, B) = (1-x)^{\frac{1}{2}} - |x|$

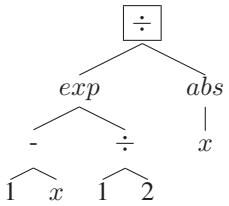

Figure 4: Mutation: $AB* = \mathsf{mutate}(AB) = \frac{\sqrt{1-x}}{|x|}$

Crossover. The crossover operation takes two individuals and produces one offspring individual and is based on subtree replacement. For example, replacing the subtree rooted at the division node (boxed ÷) in Figure 1 with the subtree rooted at the exponential node (boxed exp) in Figure 2 results in the expression tree representing AB in Figure 3: $AB = (1-x)^{\frac{1}{2}} - |x| = \sqrt{1-x} - |x|$.

Mutation. A mutation may be applied in several ways to an expression tree. For example, in Figure 3 we may change subtraction to division in AB resulting in $AB*$ in Figure 4. Mutation may have a significant impact on generated limit problems. That is, AB (Figure 3) is structurally isomorphic to $AB*$ (Figure 4), but $AB* = \frac{\sqrt{1-x}}{|x|}$ results in a more dif-

ficult limit problem at $x = 0$ due to the restrictive domain.

Computing 'Approach' Values. The second phase of limit problem construction is to determine 'approach' values (a in $\lim_{x\to a} f(x)$). Our analysis considers the domain of f. For $AB*$ in Figure 4, $\mathsf{domain}(AB*) = (0, 1]$. We intuitively consider the two endpoints of the interval as possible 'approach' values: $x = 0$ and $x = 1$. For each candidate value a, we determine the directions for which x may limit toward a: from the right ($+$), from the left ($-$), or both. For $AB* = \frac{\sqrt{1-x}}{|x|}$, we may only approach $x = 0$ from the right and $x = 1$ from the left. Hence, we generate the limit problems: $\lim_{x\to 0^+} \frac{\sqrt{1-x}}{|x|}$ and $\lim_{x\to 1^-} \frac{\sqrt{1-x}}{|x|}$.

2 Preliminaries

We restrict our definitions to the Euclidean Plane and assume the standard definition of mathematical notions such as *endpoints*, *interval*, *domain*, *function*, *continuous*, etc. We use the standard notion of a *bounded interval* which consists of all points that lie between fixed endpoints. Further, a *closed interval* contains all of its limit points.

We partition the mathematical operators into unary and binary subsets. The binary operators consist of addition ($+$), subtraction ($-$), multiplication ($*$), division ($/$ or $\div$), and exponentiation (exp or $\widehat{}$). The set of unary operators consists of basic trigonometric functions ($\sin$, $\cos$, $\tan$), absolute value ($|\ |$ or abs), limited rational powers (^{2}, ^{3}, $\sqrt{}$, $\sqrt[3]{}$, natural logarithm ($\ln$), reciprocal ($^{-1}$), and negation ($-$). Our operands are limited to a subset of the real numbers and an independent variable (commonly, x). Since coefficients do not implicitly modify the difficulty of a limit problem (while it does deepen an expression tree), we limit integer values to [0, 100] and rational numbers to those with denominators in $[2, 9]$. We also limit irrationals to π and e.

Each function corresponds to a valid expression tree in which operands are leaves and all non-leaf nodes are operators. Let f be a function with domain D. An *approach value* is a limit-point of D. An *approach side* is one of three forms depending on the domain of a function f and an approach value. Let a be an approach value for f. For an increasing (resp. decreasing) sequence of points $S \subset D$ limiting toward a we say the approach side is *from the left* (resp. right) and write a^- (resp. a^+). In cases where both decreasing and increasing sequences limit toward a in D, we say the the approach side is *two-sided*. Hence, we may formally define a single-variable limit problem.

Definition 1 (Limit Problem)**.** *A single-variable* limit problem *is a triple $\langle a, s, f\rangle$ where a is an approach value, s is an approach side, and f is single-variable function.*

For fitness of f, we compute several quantities based on the expression tree of f: number of operators t_f, number of operands r_f, the height h_f, and the maximum level-based width, w_f. While it is not our goal to echo problems from a textbook, we do look to those problems as a guide for fitness. For an initial population of problems, P_0, we compute the mean and standard deviation of each quantity. As an example, for the number of operators, we have

Algorithm 1 Function Generation

Require:
1: P_0: Corpus; GENS: Upper bound Epochs;
2: M: Mutation Rate; T: Tournament Size
3: **function** GENFUNCTIONS, (P_0, GENS)
4: $Pop \leftarrow$ INITIALIZE(P_0)
5: **for** $epoch \leftarrow 1$ **to** GENS **do**
6: $Pop' \leftarrow \emptyset$
7: **for** $j \leftarrow 1$ **to** $|Pop|$ **do**
8: $I_1, I_2 \leftarrow$ TOURN-SELECT(Pop, T)
9: $I \leftarrow$ CROSSOVER(I_1, I_2)
10: $Pop' \leftarrow Pop' \cup$ MUTATE(I)
11: **return** Pop'

$\bar{t}_{P_0} = \frac{1}{n} \cdot \sum_{f\in P_0} t_f$ and $s_t = \sqrt{\frac{\sum_{f\in P_0} t_f}{|P_0|-1}}$, respectively. We similarly compute the means $\bar{r}_{P_0}$, $\bar{h}_{P_0}$, and $\bar{w}_{P_0}$ and standard deviations s_r, s_h, and s_w.

We seek to construct limit problems that deviate from standard textbook problems; that is, we do not wish to mimic the structure of existing limit problems (by, for example, scraping textbooks). Therefore, our fitness function seeks to maximize variation in z-scores for each of the characteristics of a function f:

$$\mathsf{fit}(f) = \left|\frac{t_f-\bar{t}_{P_0}}{s_t}\right| + \left|\frac{r_f-\bar{r}_{P_0}}{s_r}\right| + \left|\frac{w_f-\bar{w}_{P_0}}{s_w}\right| + \left|\frac{h_f-\bar{h}_{P_0}}{s_h}\right|.$$

3 Algorithm

3.1 Function Generation

Our evolutionary algorithm (EA) for function generation is defined in Algorithm 1. For an initial population we use an existing corpus of textbook limit problems, P_0 (Line 4). We use a refined initial population so that we avoid iterations to search the space for a reasonable starting population.

We then construct the next generation of individuals using a standard evolutionary approach (Line 6 to Line 10). We use tournament selection (Miller and Goldberg 1995) on Line 8 to identify the two most-fit individuals (using fit) from a random subset of the input set Pop of size T by calling TOURN-SELECT. On Line 9 we then perform a CROSSOVER operation on the these two individuals resulting in a new individual I. Last, we selectively MUTATE I (Line 10) according to the mutation rate M.

Crossover. We use a 'balanced' sub-tree replacement technique as our crossover operation. That is, for expression trees S_1 and S_2 corresponding to two limit problems, I_1 and I_2, we identify respective candidate nodes n_1 and n_2 for which to perform replacement. We compute the height of a desirable substitution subtree: the difference of the minimum depth ℓ_1 of all leaves in S_1 and the height h_1 of S_1. In Figure 1, $h_d = h_1 - \ell_1 = 4 - 2 = 2$. We then choose a node n_2 with height h_d in S_2 as the root of our subtree to copy: exp in Figure 2. If no such node exists in S_2, the operation fails. We then copy S_1 with the subtree rooted at n_1 replaced with the subtree rooted at n_2; e.g., Figure 3. Since the replacement subtree is the same height as the subtree it is replacing, our operation mitigates imbalance.

Mutation. We describe three different mutations to expression trees: *substitution*, *expansion*, and *regression*.

Substitution mutation randomly selects a node and performs a syntactically correct replacement. That is, if an operator is chosen, a random, syntactically correct operator is chosen: unary or binary. If an operand is chosen, an alternate operand is generated for replacement. We restrict variable replacement such that an argument to a function (sine, square root, etc.) must maintain a variable expression: $\sqrt{x}$ will not be mutated to $\sqrt{2}$. For example, the subtraction root node of Figure 3 is substituted to division in Figure 4.

Expansion mutation selects an arbitrary node w in the expression tree of an individual and inserts a randomly-generated unary operator as the new parent of w. If the root of $AB*$ in Figure 4 is expanded to include absolute value, the expression $\left|\frac{\sqrt{1-x}}{|x|}\right|$ results; the expression tree is not depicted. Expansion may occur with a binary operator as well. In this case, we also generate a new operand.

A regression mutation severs a link in an expression tree. To be clear, regression mutation does not sever a subtree from an expression tree. If the randomly selected unary node n with parent p and single child c, the resulting subtree rooted at p would now have child node c (eliminating n). For a randomly selected binary node n with parent p and children c_L and c_R, we select the subtree with greater height: p is then the parent of c_L or c_R. For example, regressing $AB*$ in Figure 4 at the absolute value node, results in $\sqrt{1-x}/x$.

3.2 Computing Approach Values

We must generate the approach value and approach side to generate a complete limit problem $\lim_{x\to a} f(x)$. For the approach value a, we first compute $D = \mathsf{domain}(f)$, the domain, D, of f. For meaningful limit problems, candidate approach values include finite interval endpoints in D (e.g., $x = 0$ for $\sqrt{x}$), around discontinuities (e.g., $x = 0$ for $(x-1)/x$), and end behavior (e.g., $x \to \pm\infty$). We also consider as candidates the endpoints of subintervals with piecewise defined functions (e.g., $x = 1$ in $|x-1|/x$).

For f and each candidate approach value a, we approximate the limit infimum and limit supremum numerically from the left and/or right. We do so by constructing an *approach sequence* of domain values of size M. We then add subsequences of size N between each value in the approach sequence. Last, we evaluate f at each value in the approach sequence using an expert system (Wolfram and Gray 2018). We can approximate a limit infimum and limit supremum using this codomain sequence.

For end-behavior approach values $a = -\infty$ or $a = \infty$, we expand the distance between subsequent points as x approaches a. That is, $\{x_i\}$ is constructed such that for $1 \le i \le M$, $|x_{i+1}/x_i| > K_\infty > 1$ for some constant K_∞. For $a = -\infty$, it is clear $x_i < 0$ for all i; similarly, for $a = \infty$, $x_i > 0$ for all i. In the case of a finite approach value a, we consistently shrink the distance between subsequent points such that for $1 \le i \le M$, $|a - x_i| > |a - x_{i+1}| > 0$. If we approach a from the right, for all i, $x_i > a$; similarly, approaching a from the left requires $x_i < a$ for all i.

Given an approach sequence of values, we construct a set of N interior sequences such that each element is between the values in the approach sequence $\{x_i\}$. That is, each interior sequence $\{z_j\}$ of size N is generated such that for all $1 \le i \le M-1$, $x_i < z_j < x_{i+1}$ for all $1 \le j \le N$. We then combine these x-values into a large, *ordered* approach sequence: $X = \{x_i\} \cup \{z_j\}_1 \cup \ldots \cup \{z_j\}_{M-1}$. We then compute the corresponding sequence of function values: $F_X = \{f(x) \mid \forall x \in X\}$.

We then approximate the limit infimum and limit supremum. If, as x approaches a in X, F_X numerically limits to a finite value L, the limit exists. We thus construct the associated limit expression noting a and the approach side. So for $a = \infty$, we might write $\lim_{x\to\infty} f(x) = L$ or in the case of a finite a from the right, we may write $\lim_{x\to a^+} f(x) = L$.

If the limit of F_X is numerically inconclusive, we approximate $\limsup_{x\to\infty} f(x)$ by taking maxima of progressive subsequences of F_X; similarly for $\liminf_{x\to\infty} f(x)$ and minima. If the limit infimum and limit supremum computations imply F_X is monotonic, we know end-behavior limits tend toward $\pm\infty$. By definition, if $\liminf_{x\to a^-} f(x) = \limsup_{x\to a^-} f(x)$ then $\lim_{x\to a^-} f(x)$; similarly for $\lim_{x\to a^+} f(x)$. Last, if $\lim_{x\to a^-} f(x) = \lim_{x\to a^+} f(x) = L$ it follows $\lim_{x\to a} f(x) = L$. We generate limit problems accordingly.

Table 1: Characteristics of Limit Problem Corpus

	Operators	Operands	Height	Width
Mean	3.79	3.69	4.03	2.85
Std. Dev.	1.60	1.64	1.06	1.07

4 Simulations

Setup. Our corpus of input limit problems consisted of 37 limit problems from a seminal Calculus textbook (Larson, Hostetler, and Edwards 2002). We use Mathematica (Wolfram and Gray 2018) as an expert system for computing descriptive elements of each function (e.g., domain, etc.).

We ran the algorithm described in §3 with the following parameters. Selection uses tournament selection with tournament size 5 and elitism (the unaltered most-fit individual always proceeds to the next generation). To avoid expression tree bloat, we used tree size (tree node count) 20 as the maximum number allowed after the crossover operation. After some tuning, the mutation rate for each type of mutation was set to 0.1. If regression would result in a tree of size smaller than the minimum tree size of 5, no regression occurs.

A main focus of our approach is ensuring that the search space is free of fundamentally uninteresting or unreasonably difficult expressions. In our implementation we exclude the following configurations of binary operators, unary operators, numbers, and variables: simplifiable constants (e.g., $2+2$), trivial unary operations such as $|2|$, and complex variable expressions (x^x, $2^{\sin x}$, x^{e^x}, $(x^3)^x$, $\cos x^{\sin x}$).

Results. We present some of our preliminary results. In order to greater appreciate our results, we describe a sample problem from our corpus. Many textbook limit problems like $\lim_{x\to 2^+}(x-3)/(x-2)$ evidence a domain restriction

Table 2: Generated Limit Problems after 100 Generations

$\lim_{x\to\ln 3^-} \frac{\lvert x\rvert + 9}{(3-e^x)(x-5)x}$	$\lim_{x\to 0} \frac{3e^e\lvert x\rvert}{5x^2}$
$\lim_{x\to 5^-} \frac{9}{(3-e^e)(x-5)x\lvert x\rvert}$	$\lim_{x\to 0^+} \frac{(-x-4)x-2x+5}{-\frac{x}{e}-2x}$
$\lim_{x\to 5^-} \frac{9\lvert x\rvert}{(3-e^x)(x-5)x}$	$\lim_{x\to\ln 3^-} \frac{3\lvert x\rvert}{25(3-e^x)(x-5)}$
$\lim_{x\to\ln 3^+} \frac{9\lvert x\rvert - x}{(3-e^x)(x-5)}$	$\lim_{x\to 5^-} \frac{9\lvert x\rvert - x}{(3-e^x)(x-5)}$

and thus are quite straightforward in terms of their solving. This simplicity is quantitatively evident in the number of operators (3), operands (4), height (2), and width (4). We can see this general trend of 'simpler' limit problem functions in our corpus by observing the mean and standard deviations in Table 1.

Our goal is not to mimic textbook problems, but construct more diverse and complex limit problem templates as described in (Singh, Gulwani, and Rajamani 2012). We report the means for generated problems contrasting our corpus problems (Table 1): operators (10.65), operands (10.69), height (6.88), width (7.69), and fitness score (15.59). Recall that our limit problem fitness function (§2) maximizes combined z-scores. Our upper bounds, in concert with our fitness function thus shrink the search space to more palatable problems. We present a sample of problems constructed after 100 generations in Table 2. Compared to the simplicity of the corpus problem $\lim_{x\to 2^+}(x-3)/(x-2)$, our generated problems are much more complex, yet manageable due to our upper bounds on expression tree characteristics.

Discussion. We believe our algorithm provides a solid framework for limit problem and limit problem template generation; however, further tuning of parameters should be investigated. Specifically, our fitness functions can be moderated according to the desired problems. For example, if we wish to explore the space of textbook problems, we would desire fitness scores of 0 which indicates an 'average' problem. In fact, in some early experiments we are able to generate simple textbook problems such as $\lim_{x\to 3^-}(x-2)/(x-3)$.

5 Related Work

Koza (Koza 1996) provides a comprehensive view of genetic algorithms and genetic programs. In section 7.3, Koza provides an example of algebraic expression manipulation using an EA. He explores the problem of symbolic regression using a set of functions similar to the set used in our simulations. Although our goals differ, Koza explores a selection procedure based on a "fitness roulette wheel." In this roulette implementation, Koza assigns each individual a proportion of a roulette wheel based on that individual's fitness value compared to the combined fitness values of all individuals in the population. When the roulette wheel is 'spun,' the individual with the highest fitness value has the highest probability of being selected for a given position in the next generation's population. This contrasts our tournament selection procedure of most-fit individuals in a random subset.

(Ferreira 2001) describes an alternative representation to standard genetic algorithms by introducing *gene expression programming*. This technique defines an individual to be a fixed-length string composed of elements from two alphabets: (1) a set of functions ($+$, $-$, etc.) and terminals (variables and constants) and (2) a set of terminals. The manipulations Ferreira defines on the strings guarantees syntactically valid expressions. This technique is of interest if we wished to explore the space of template functions for 'simpler' limit problems. However, we are left unconvinced that the entire space could be explored with Ferreira's algorithms. We chose not to mimic the technique of Ferreira due to the use of fixed-length strings. We believe non-expansive strings would stifle the search for challenging function templates for limit problems.

Grosan (Grosan 2004) follows Ferreira by evolving expressions as fixed length strings of operators and operands. Their fitness function takes the minimum of all constituent subexpressions encoded in an individual justifying their choice as "neither practical nor theoretical evidence that one of these expressions is better than the others." Whereas, our technique attempts to define fitness based on diversity of function characteristics.

6 Conclusions

We described a two-step process to limit problem generation. First, we defined an evolutionary approach to function generation via syntactic tree manipulation. We believe this general approach can be used to generate expressions for other classes of problems such as arithmetic expression simplification (with $+$, $-$, $*$, $\div$), domain computation, complex derivatives, and more. Our second contribution is limit problem construction via systematic numerical analysis of function domains. Our experiments indicate the generated problems are more appropriate as challenge problems for students. Although our results are preliminary, we feel constructing diverse problem templates may be a tool for student mastery and may aid educators when writing homework sets or difficult exam problems.

References

Ferreira, C. 2001. Gene expression programming. *Complex Systems* 13(2):87–129.

Grosan, C. 2004. Evolving mathmatical expressions using genetic algorithms. In *Genetic and Evolutionary Computation Conference (GECCO)*.

Koza, J. R. 1996. *Genetic programming*. Cambridge, MA: MIT Press.

Larson, R.; Hostetler, R.; and Edwards, B. 2002. *Calculus with analytic geometry, 3rd Edition*. Houghton Mifflin.

Miller, B. L., and Goldberg, D. E. 1995. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems* 9(3).

Singh, R.; Gulwani, S.; and Rajamani, S. K. 2012. Automatically generating algebra problems. In *AAAI*.

Wolfram, S., and Gray, T. 2018. Wolfram mathematica student edition version 10.