Evolutionary algorithms

Orsolya Sáfár

Genetic programming

Genetic programming is a class of evolutionary algorithms where solutions have a tree graph representation.

The tree structure is a natural representation for arithmetic expressions, logical expressions or even entire program codes.

Some curve fitting problems and decision trees also have tree representation for the solutions.

Prefix form

In the prefix form of an expression, the operator is written first, followed by the operands between brackets. For example, consider the expression

$$\sin\left(\frac{xy}{x+y+1}\right)$$

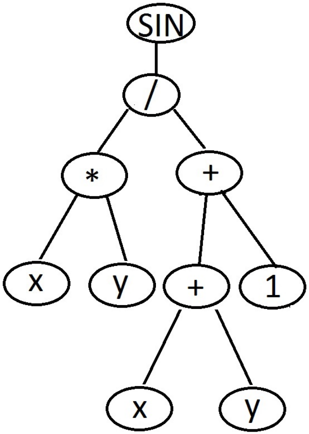
. The corresponding prefix form is:

$$\sin(/(*(x,y),+(+(x,y),1)))$$

Prefix form is easy to rewrite in a tree structure. Vertices correspond to operators, and each vertex has a number of children equal to the operands of the operator. For non-commutative operators, the order of the children is relevant.

Example

$$\sin(/(*(x, y), +(+(x, y), 1)))$$



Tree representation

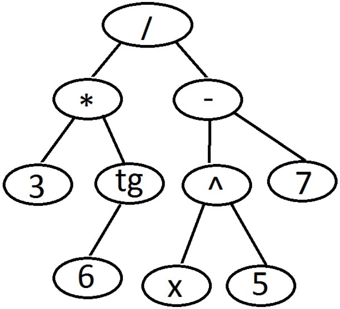
Two types of vertices may be used in a tree:

- operators correspond to inside vertices (vertices with children)
- operands correspond to leaves (vertices without children)

From the tree, the prefix form can be reconstructed with an exhaustive depth-first search. Starting from the root of the tree, we go to the leftmost children for as long as possible. After reaching a leaf, we take a step back, and turn right at the first possibility. Then repeat. The prefix form is rewritten according to the order of the vertices of the search.

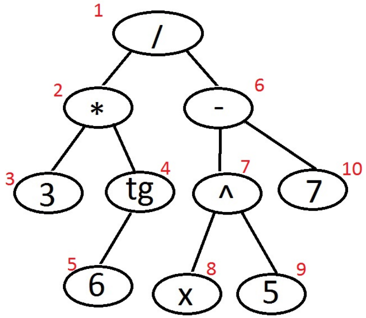
Example

Determine the expression represented by the following tree.



Example

First we run the depth-first search.



Then the prefix form is: $/(*(3, \mathrm{tg}(6)), -(\wedge(x, 5), 7))$, so:

$$\frac{3 \cdot \mathrm{tg} 6}{x^5 - 7}$$

Logical expressions

Logical expressions can be represented similarly. For example, try to represent the following expression as a tree:

$$(x \land \mathit{true}) \rightarrow ((x \lor y) \lor (z \leftrightarrow (x \land y)))$$

It can be an issue when the solutions need both logical and arithmetic expressions. For example,

$$(N > 30) \lor (D < 20 \land M \geq 100)$$

is a proper expression, but $N \land 100$ is meaningless. To construct proper trees, we need to make further restrictions.

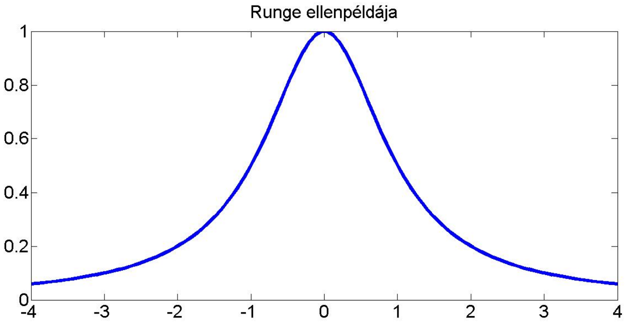
Example: symbolic regression

A set of functions is given, e.g. basic arithmetic operations, and $\sin, \cos, \exp, \mathrm{abs}$ functions. Our goal is to approximate an unknown function with a composition of these. The value of the unknown function is known in some (e.g. 20) points. Then the previous functions are the operators, and $x \cup \mathbb{R}$ are the operands.

The problem is not defined properly at this point. It is known that any function known at n points can be approximated by a polynomial of degree $n - 1$. So there exists a polynomial of degree 19 that takes exactly the given values at the 20 known points, but this is not the function we are looking for, because this interpolation has otherwise poor properties.

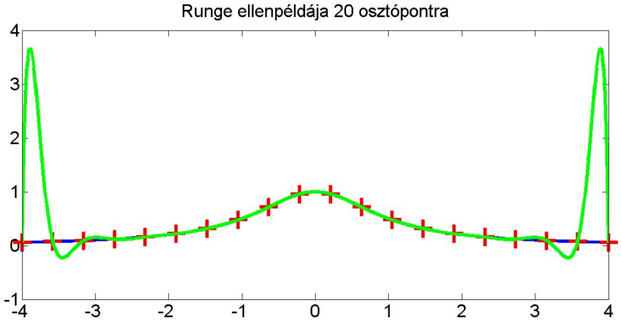
Runge's counterexample

Take the function $f(x) = \frac{1}{1+x^2}$. We watn to appoximate it on $[-4, 4]$. Take equidistant points for base points (where the value is known), then calculate the unique interpolating polynomial (Lagrange polynomial).



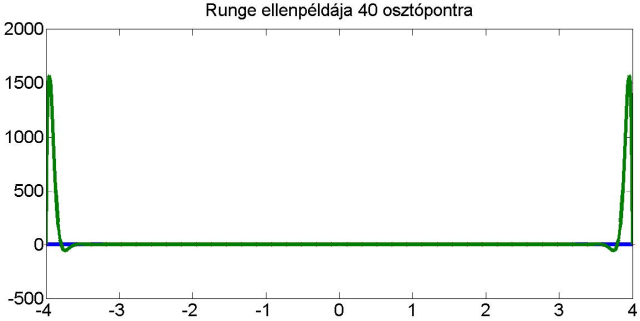
Runge's counterexample

The interpolating polynomial of degree 19:



Runge's counterexample

The interpolating polynomial of degree 39:



The high degree polynomial approximation oscillates wildly near the endpoints of the interval, which is not good in general. Instead, we are looking for a "short" expression. The length of the expression is controlled during the algorithm to make it short.

Construction of the initial population

First we determine a maximal depth of the expression. Then we use either of the following methods to create the tree:

- Every branch has length equal to the maximal depth. Inner vertices are uniformly random among operators, and leaves are uniform among operands
- Starting from the root, every vertex is selected randomly from the set of operands and operators. If we reach the maximal depth, we select randomly from operands only.

Crossover and mutation

Unlike the majority of genetic algorithms, genetic programming does not use both operators. It uses only one of them, selected randomly.

Mutation is the connection of a new random subtree (branch) to a random vertex. If the vertex is an inner vertex, the original subtree from that vertex is discarded.

For crossover, we select two vertices randomly at each parent, and we swap (exchange) the subtrees at those points.

Population size

Both maximising fitness and mutation in general causes the trees to become larger and larger (bloating). To control this, we either introduce a maximal depth, or apply a penalty function for trees too big.

Since we typically work with large populations, the selection pressure is very strong. To still give a reasonable chance of survival for lower fitness solutions, we apply the 80/20 rule: 80% of parents are selected from among high fitness solutions, and 20% are selected from among low fitness solutions. The distinction between "high" and "low" depends on the population size.

Application: optimal compression

A 3-dimensional surface is given as a function over a 2-dimensional lattice. Our goal is to select some the lattice points and find an interpolation of the original surface using interpolation over the selected points.

The ratio of lattice points that can be selected is a priori given (between 1% and 10%). The question is how to select them. Design a genetic algorithm for this problem.

One of the main questions is how to measure the fitness of the set of selected points. We define the fitness function using angle-optimal triangulations.

Optimal triangulation

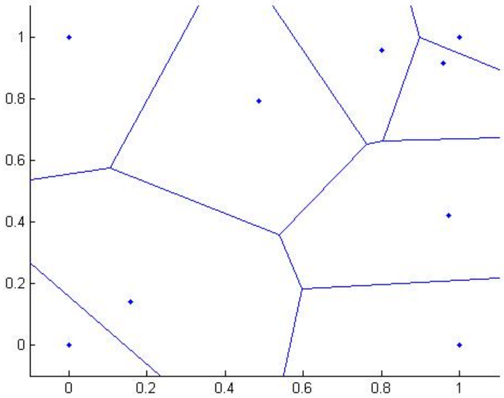
To approximate the function, we use a function that is linear over certain domeains, where the domains are triangles in the triangulation of the lattice points.

There is an exponential number of triangulations, since for any 4 points in a convex position, we can draw the diagonal in 2 ways, and 5 points in a general position always contain at least 4 points in a convex position.

We are looking for a triangulation that does not contain very small angles (which correspond to long triangles). For each triangulation, we take the list of all the angles in increasing order. From these lists, we consider the largest in lexicographic ordering to be optimal (this may not be unique). We need some preparations.

Voronoi–cells

For a given finite set of points, the Voronoi-cell belonging to point P is the set of points which are closer to P than to any other point in the given set.

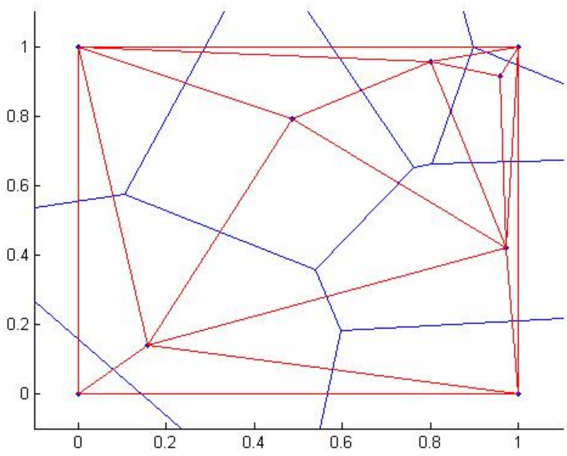


Delaunay–triangulation

The Delaunay–triangulation is the graph dual of the Voronoi–cells (we do not prove it).

The dual graph is the following: cells correspond to vertices of the dual graph, and two vertices are connected if the corresponding cells are adjacent. This is called the Delaunay–triangulation.

Delanuay–triangulation



Delanuay–triangulation

If the number of the given points is n and the convex hull contains k points, then the triangulation consists of $2n - 2 - k$ triangles.

Proof Every edge belongs to 2 triangles. Inner triangles have 3 edges, and the outside domain has k edges. The triangulation is a planar graph, so Euler's formula ($n - e + f = 2$, where e is the number of edges and f is the number of domains (faces)) applies:

$$n - \frac{3(f-1)+k}{2} + f = 2$$

from where

$$2n - 3f + 3 - k + 2f = 2$$

and solve for f.

Fitness function

The original surface is approximated by triangles in 3 dimensions: the vertices are determined by the Delanuay–triangulation, with the third coordinates determined by the value of the function on the vertices of the triangles.

We calculate the sum of the absolute value of the difference of the original function and the approximation at the lattice points. The Delanuay–triangulation ensures that the difference is 0 on vertices of the triangles.

The sum gives the total error of the approximation. This is the raw fitness, which we want to minimize.

Genetic operators

Initial points

We want to avoid irregular shapes as much as possible, so instead of a completely random selection of the initial k points, so we take $\frac{\sqrt{k}}{2}$ on the edge of the lattice, and the rest randomly.

Crossover

We choose an axis randomly, then a constant value on this axis. We divide both set of points into two parts along this value, then join the two halves along this value (similar to one-point crossover). Then we need to correct the number of points by removing or adding points randomly to get to exactly k points.

Mutation

We replace one point in the set by an adjacent point.