Genetic Programming on IXM

Eric Schulte

13 December 2009

Report

Boards

quick desc. and properties relevant to GP

GP

implementation and how/why different from vanilla GP

evo co-evo

results

Data Analysis

basic GP parameters

just to show that mutation and crossover actually help, and to justify the choices used in later experiments

GP visualization

evo-individuals and coevo-individuals
connect the gnuplot graphics to the text files of results

1. ingest text files
2. persist in serialized ruby structures
3. connect to group/board data structures
4. produce graphs

plot fitness by time

many data points, maybe a best-fit line w/R

comparisons

sharing rates

```
ave_max_time = {}
[100, 1000, 10000].each do |share|
  data_s = goal2.select{|d| d.share == share}
  ave_max_time[share] = (0..9).map{ |run| data_s.sort_by{|d| d.time}.last }.
    inject(0){|a, p| a += p.time } /␣4
end
```

need to clear out individuals from previous runs – namely those returned before the reset packet

```
# make sure to remove individuals from before reset packet
temp_goal2 = goal2.reject{|d| d.time < 4};

ave_best_score = {}
[100, 1000, 10000].each do |share|
  data_s = temp_goal2.select{|d| d.share == share}
  ave_best_score[share] = (0..9).map{|run| data_s.sort_by{|d| d.score}.first}.
    inject(0){|a, p| a += p.score } /␣9
end

best_inds = {}
[100, 1000, 10000].each do |share|
  data_s = temp_goal2.select{|d| d.share == share}
  best_inds[share] = (0..9).map{|run| data_s.sort_by{|d| d.score}.first}.
    sort_by{|d| d.score}.first
end
```

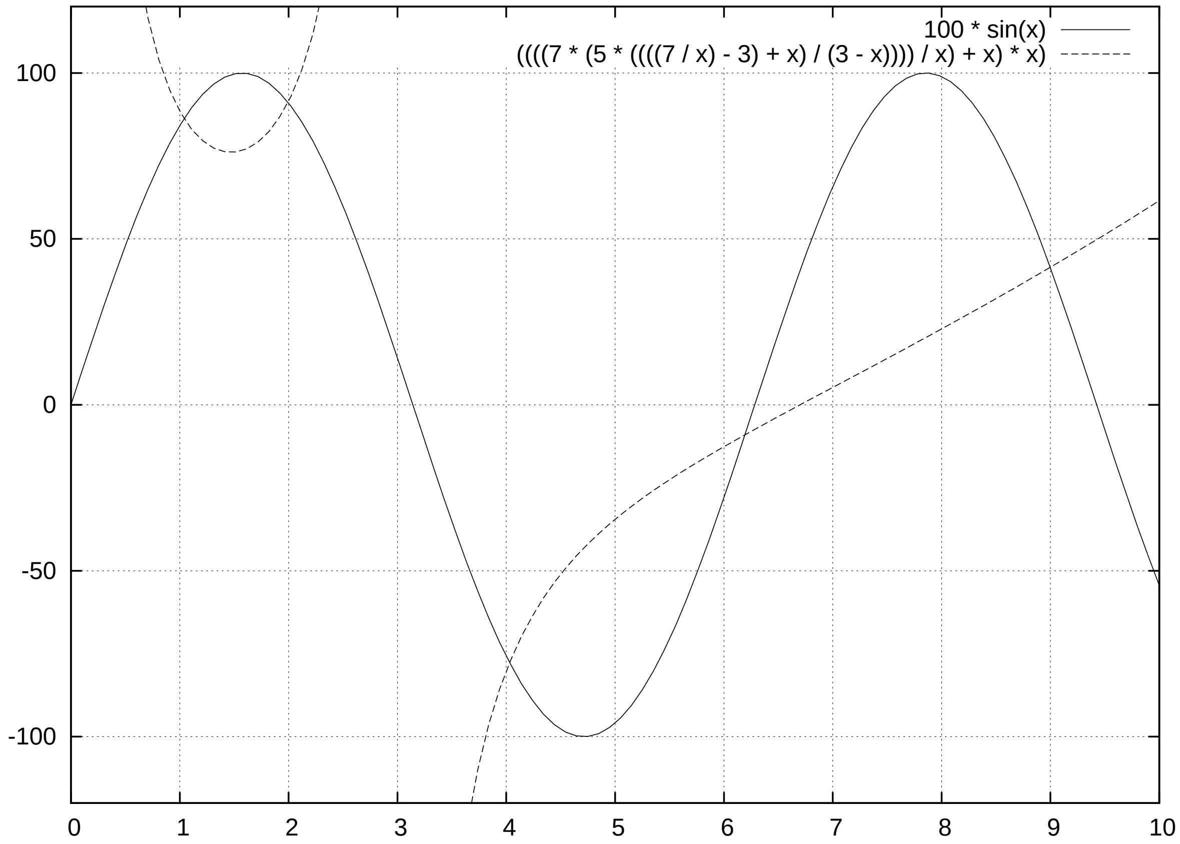
- Goal 2
  - runtime – no runs completed

    ```
    irb(main):052:0> ave_max_time
    ave_max_time
    {10000=>1207.943955, 100=>1207.720609, 1000=>1210.959188}
    ```

  - score – looks like two actually succeeded...

    ```
    irb(main):096:0> ave_best_score
    ave_best_score
    {10000=>255.311111111111, 100=>253.433333333333, 1000=>183.966666666667}
    ```

  - best individuals



line vs. eight

common tools

ingest

ingest a directories worth of run results and return a list of Datum

```
class Datum
  attr_accessor :share, :goal, :run, :time, :score, :path
end
def ingest(base)
  Dir.entries(base).map do |e|
    if e.match(/r_s.(\d+)_m.(\d+)_b.(\d+)_i.(\d+)_g.(\d+).(\d+)/)
      share = Integer($1)
      goal  = Integer($5)
      run   = Integer($6)
      File.read(File.join(base, e)).map do |l|
        if l.match(/^([\d\.\/-]+)\t([\d\.\/-]+)\t([frl]+)$/)
          d = Datum.new
          d.share = share
          d.goal  = goal
          d.run   = run
          d.time  = Float($1) rescue -1
          d.score = Float($2) rescue -1
          d.path  = $3
          d
        end
      end.compact
    end
  end.compact.flatten
end
```

test ingest – works – 512783 data points in the directory

```
<<ingest>>
data = ingest("./raw/15-evo-eight/");''
puts data.size
```

- serialize – not plausible
  tried YAML and sqlite3 and neither worked in a reasonable amount of time

  creating a sqlite3 table to hold this info

  ```
  # create database
  db = SQLite3::Database.new('raw.db')

  table = "evo_eight"

  # create table
  db.execute("create␣table␣#{table}␣(share␣INT,␣goal␣INT,␣run␣INT,␣time␣FLOAT,␣s

  # define keys
  keys = %w{share goal run time score path}

  # create a large insert statement for 1000 data points
  stmt = data.map{ |d| "insert␣into␣#{table}␣(#{keys.join(", ")})␣values␣(#{keys

  db.transaction{ |db| db.execute_batch(stmt.join("\n")) }
  ```

rpn to alg

evo individuals are check on the (0..9) range inclusive

0757x/3-x+3x-/**x/x+x*

```
operators = %W{+ - /␣*}
$stack␣=␣[]
ind[0][0].split(//).each␣do␣|ch|
```

```
␣␣if␣operators.include?(ch)
␣␣␣␣right␣=␣$stack.pop␣or␣"1"
␣␣␣␣left␣=␣$stack.pop␣or␣"1"
␣␣␣␣$stack.push("(#{left}␣#{ch}␣#{right})")
␣␣else
␣␣␣␣$stack.push(ch)
␣␣end
end
puts␣$stack.pop
```

```
((((7 * (5 * ((((7 / x) - 3) + x) / (3 - x)))) / x) + x) * x)
```