CyberTiggyr Evie: Genetic Algorithms for Lisp

Gene Michael Stover

created 2001 October
updated Sunday, 2006 March 26

Contents

1 Introduction

2 License

Here is a copy of the license agremeent that is on the source code. The same agreement appears in each source code file.

3 Obtaining

There are two Lisp source files. The only one you need is evie.lisp. The other file, loadall.lisp, is optional & wouldn't exist if I would get around to doing the "learn to use ASDF" item on my To Do list.

- http://cybertiggyr.com/gene/evie/src/evie.lisp
- http://cybertiggyr.com/gene/evie/src/loadall.lisp

4 About Genetic Algorithms

For an introduction to genetic algorithms, see [1].

Here are some of the rules & assumptions about genetic algorithms in Evieworld.

4.1 Population

In Evie, a population is always a proper list of organisms.

4.2 Organisms

To Evie, an organism is opaque data. The client of the Evie library supplies functions to make new children (i.e., mate parents) & to obtain fitness values for organisms.

4.3 Fitness values

In Evie, fitness values are always scalar numbers which may be compared with the Common Lisp function `#'>`. Examples of fitness values include

- integers, such as -101 and 17,
- real numbers, such as 22.22 and 31.415927, or
- ratios, such as 22/7 and the result of `(/ 1 3)`.

Larger fitness values *always* indicate organisms which are better than organisms with smaller fitness values[1]. For example, if organism A has a fitness of 10, & organism B has fitness of -101, then Evie assumes organism A is better than organism B.

From Evie's point of view, fitness values are dimensionless[2]. The only purpose of a fitness value is to be compared to the fitness values of other organisms in the same population & same generation.

5 Example: System of equations

Given this system of three equations:

fixme

$x^2 + 2y^2 + 3z^2 = 116$
$4x^2 + 5y^2 + 6z^2 = 266$
$7x^2 + 8y^2 + 9z^2 = 416$

let's use a genetic algorithm to solve for x, y, & z.

The system happens to have a solution at $x = 3$, $y = 4$, $z = 5$, but our genetic algorithm doesn't know that.

Our genetic algorithm will not make use of some of Evie's higher-level features. Normally, I think it's best if a first example shows the highest level, easiest use of a library, but in this case & for some reason I don't understand, I think it's better to show a simple, lower-level use first. I guess I think it's good to see what's happening at a simple level first so we can understand the benefits of the higher-level features later.

The first thing to do is to decide on a representation of the organisms which will be points in the problem space. We must decide how to decode them into

[1] If smaller values were better, the attribute would be cost, not fitness.

[2] In fact, fitness vales are inherently dimensionless in any genetic algorithm unless the programmer takes special steps to ensure otherwise, but such special measures are, well, lame.

solutions, derive fitness values from them so we can compare them, & combine two parent solutions to make a child solution.

We'll pretend we know nothing about the solution at $x = 3$, $y = 4$, $z = 5$. We'll pretend that we don't even have a good idea of the range in which a solution might be found. So we'll use a numeric representation with a wide range. What numeric representation has a wide range? Floating point. In other words, an organism is a structure with three floating point values: x, y, & z.

```
(defstruct ex01-organism
  xbits                                 ; list of 16 bits
  ybits                                 ; list of 16 bits
  zbits)                                ; list of 16 bits
```

The three members of EX01-ORGANISM are bit strings. Each one encodes one of the x, y, or z components of 3-space. We'll need to convert them to numbers to plug them into the original equations. Here are some functions which do that. They use Evie's DECODE-BITS-AS-FIXED function for convenience.

```
(defun ex01-organism-x (organism)
  (decode-bits-as-fixed (ex01-organism-xbits organism) 16.0))

(defun ex01-organism-y (organism)
  (decode-bits-as-fixed (ex01-organism-ybits organism) 16.0))

(defun ex01-organism-z (organism)
  (decode-bits-as-fixed (ex01-organism-zbits organism) 16.0))
```

By calling DECODE-BITS-AS-FIXED with a SCALE argument of 16.0, we are effectively placing 4 bits of the bit string behind the binary point. Also, since 16.0 is a floating point number, the result from DECODE-BITS-AS-FIXED will be floating point.

To figure the fitness value for an organism, we apply it's three values to the system of equations, producing three new values. We take that point's distance from the right-hand sides of the original three equations. The farther the new point is from the original point, the worse the organism is. Remember that in Evie, larger values indicate organisms which are more fit, so we must negate the distance. So larger distances will produce more negative fitness values which Evie will interpret as less-fit organisms.

Here is the fitness function:

```
(defun ex01-fitness (organism)
  (let ((x (ex01-organism-x organism))
        (y (ex01-organism-y organism))
        (z (ex01-organism-z organism)))
    (let ((d0 (- 116 (* x x) (* 2 y y) (* 3 z z)))
          (d1 (- 266 (* 4 x x) (* 5 y y) (* 6 z z)))
          (d2 (- 416 (* 7 x x) (* 8 y y) (* 9 z z))))
```

```
      ;; Strictly speaking, the distance between the original
      ;; three right-hand sides of the equations & the values
      ;; we get from the organism would be the square root of
      ;; the sum of the squares of the differences, but that
      ;; outer square root isn't necessary.  So we can shave
      ;; a tiny bit of computation time by omitting it.
      ;; We must remember to negate the sum so that smaller
      ;; distances become better fitnesses.
      (- 0.0 (* d0 d0) (* d1 d1) (* d2 d2)))))
\end{vebatim}

Here are some examples of that fitness function
in action:

\begin{verbatim}
;; Not a very good solution
(ex01-fitness (make-ex01-organism :xbits '(1)
                                  :ybits '(1)
                                  :zbits '(1)))
=> -257267.53

;; Better
(ex01-fitness (make-ex01-organism
                :xbits '(1 0 0 0 0 0 0 0 0 0)
                :ybits '(1 1 0 0 0 0 0 0 0 0)
                :zbits '(1 1 0 0 0 0 0 0 0 0 0)))
=> -381.0

;; Best
(ex01-fitness (make-ex01-organism
                :xbits '(1 1 0 0 0 0 0 0 0 0)
                :ybits '(1 0 0 0 0 0 0 0 0 0 0)
                :zbits '(1 0 1 0 0 0 0 0 0 0 0)))
=> 0.0
```

Besides defining our organism type & a fitness function for it, we must write a function a new, child organism from two parent organisms. Not surprisingly, this is called mating.

A critical component of making a new child organism is *crossover*. Crossover is the process (or effect?) by which the child inherits genetic material from both parents. There are many type of crossover, but the basic, common one, which tends to be adequate for problem-solving in practice, is single-point crossover. Evie provides a single-point crossover function for convenience; it is called CROSSOVER1.

```
(defun ex01-organism-mate (a b)
```

```
  "Return a new organism created by a crossover
operation between A & B. A & B are organisms."
  (make-ex01-organism
   :xbits (crossover1 (eth01-organism-xbits a)
      (eth01-organism-xbits b))
   :ybits (crossover1 (eth01-organism-ybits a)
      (eth01-organism-ybits b))
   :zbits (crossover1 (eth01-organism-zbits a)
      (eth01-organism-zbits b))))
```

Wow, that was a lot of code for a simple example. At least it's all simple code.

Now let's

fixme

6 API Summary

Here are lists of Evie functions grouped by their uses.

loops **evie-loop** Basic loop. Uses a mating schedule rather than a fitness function. Section **??**

evie-fit-loop Basic loop. Uses a fitness function rather than a mating schedule. Section **??**

evie-tier-loop An experimental loop that tries to preserve diversity by recursively restarting itself & merging the results. Somewhat effective. Section **??**

scheduling functions These functions implement different mating schedules.

ordinal-schedule Section **??**

tournament-schedule Section **??**

operations on populations These functions exist for convenience. They perform operations on populations. Populations in Evie are always lists of organisms.

diversity Return the number of distinct organisms in a population. If two organisms have the same genetic code, they are not distinct. If all organisms in the population are different, it'll return the length of the population. On the other hand, if the population has lost diversity, it'll return 1. Compares organisms with EQUALP .

sort-by-fitness Given a list of organisms, return a new list of the same organisms which is sorted from best fitness to worst. Section **??**

best-of-population Given a population & a fitness function, return the organism with the largest fitness value from the population. Section **??**

bit vectors These functions perform operations on bit vectors which are commonly necessary in genetic algorithms. There's no need to use them, but they are often convenient if your genetic encoding uses bit vectors.

random-bits Given a length, return a bit vector with that length, initialized with random bits. Section **??**

crossover1 Perform single-point crossover on two bit vectors, returning a new bit vector. Section **??**

mutate-reverse Given an organism, always mutate it & return a new organism. Supports only one mutation: reversal ofa randmly select section of the organism. Section **??**

decode-bits-as-fixed Makes it easy to convert a bit string of genetic material into a scalar, probably to use it in a fitness function. Section **??**

misc helpers next-generation Deprecated. Given a list of organisms, a fitness function, & a mating function, return a new list of child organisms. Uses ordinal scheduling. Does not sort the new list. Section **??**

tournament Used by TOURNAMENT-SCHEDULE. Section **??**

7 Reference

A Hand-written notes

I scanned some of my hand-written notes into a PDF. They are in http://cybertiggyr.com/gene/evie/notes000.pd

B Other File Formats

- This document is available in multi-file HTML format at http://cybertiggyr.com/gene/evie/.
- This document is available in Pointless Document Format (PDF) at http://cybertiggyr.com/gene/evie/evie.pdf.

References

[1] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, October 1989.