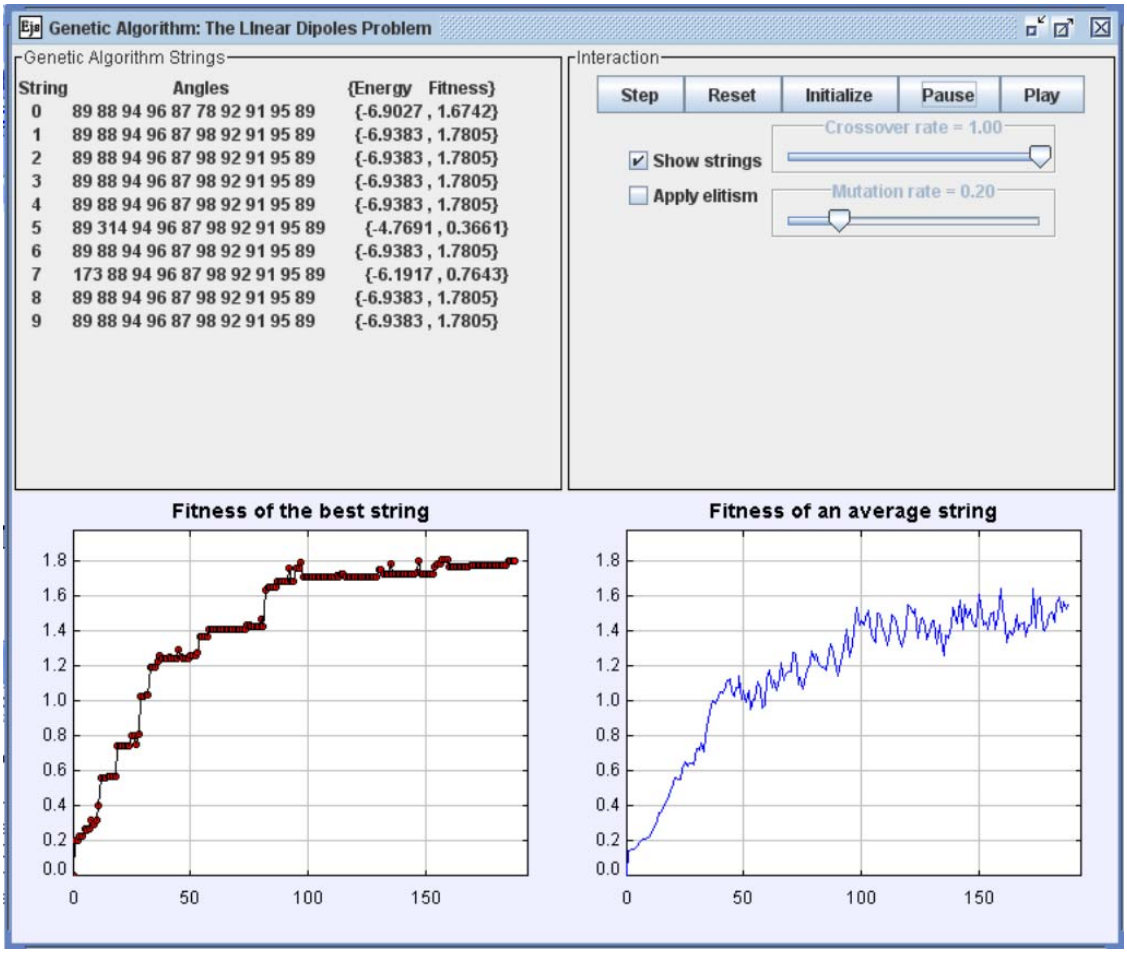
Genetic Algorithm Linear Dipoles Program

The simple Genetic Algorithm (GA) sin curve fitter can be run using Ejs (see the Ejs ReadMe file if you have not yet installed Ejs).



The aim of this GA is to find the minimum energy for a set of small identical dipoles arranged in a straight line. The dipoles are pinned at their midpoint an equal distance apart, but can rotate around this point. In the optimum arrangement, all dipoles are at an angle of 90^{o} to the vertical.

To aid display, the GA runs with a relatively small population of just ten strings. These strings, their energy and associated fitness are displayed in the top left of the window. The variation with generation number of the fitness of the best string in the current population is displayed to the lower left and the fitness of the average string in the population is shown to the lower right.

BUTTONS

Button	Function
Step	Run the program one cycle at a time
Reset	Reset all parameters to their initial values and start the program from the beginning.
Initialize	Carry out a **Reset**, but do not clear the charts showing progress of the previous calculation. This allows one to compare the results of two runs using identical parameters.
Pause	Temporarily halt execution
Play	Restart execution

SLIDERS

Slider	Function
Crossover rate	Variable between 0 and 1.0
Mutation rate	Variable between 0 and 1.0

TICK BOXES

Tick box	Function
Show strings	Toggles the display of the strings
Elitism	Toggles use of elitism by the GA

Investigations & Exercises

1. Default parameters

Run the algorithm using the default parameters. Note the fairly organized rise in the fitness of the best string and the noisier appearance of the plot of average fitness.

2. The influence of random numbers

Run the algorithm for a few hundred cycles using the default parameter settings. Press the **Initialize** button, which will restart the calculation from scratch, but will leave the fitness traces from the previous run displayed. Note that the behavior of the algorithm is the same in general terms in each new run but that, because the algorithm relies upon random numbers, successive runs are never identical.

3. Crossover

Reset the simulation and allow it to run for several hundred generations. Press the **Initialize** button and immediately reduce the crossover rate to 0. You should find that the performance is somewhat worse, but that the algorithm still makes some progress towards the optimum solution. Because the population is small the cycles pass quickly. In each cycle, the mutation rate is quite high so the strings continue to adjust, though progress is usually at a slower rate than when crossover operates.

4. Mutation

Press the **Reset** button, then reduce the mutation rate to zero. Evolution ceases almost immediately as the population of strings becomes full of identical strings. (You may notice that the fitness of the average string shows occasional small blips. This is because the slider linked to mutation reduces the mutation rate not to zero, but to a very small value. You can demonstrate this by going into the View panel of Ejs and changing the lower limit of the mutation slider from 0 to, say -1.0 and rerunning the [program, moving the mutation rate to a value of less than zero.)

5. A dynamic look at mutation

Reset the calculation and then move the mutation slider, watching the effect that different mutation rates have on both the fitness of the best string and of the average string. Note that while a very low mutation rate can lead to progress towards an optimum solution almost coming to a halt, a very high mutation rate is destructive and may prevent the algorithm converging.

6. Elitism

Perform runs with elitism selected through the appropriate tick box. Note the effect on the fitness of the best string, which now never falls. You will also see that the first string in the list shown at the top left changes only rarely, as the best string (string 0) changes only when it is displaced by one of higher quality.