

CS 495/595: Evolutionary Computation
Lecture notes for January 23, 2008
Introduction, Programs for Evolutionary Computation
Alden H. Wright, Department of Computer Science
University of Montana, Missoula, MT 59812
alden.wright@umontana.edu

Reading: Chapter 1 of the Ashlock text.

1 Introduction

Evolutionary computation is an example of biologically inspired computation. In other words, the idea of evolution is used to structure an algorithm. Solutions to the problem at hand are represented as data structures, and these data structures undergo simulated evolution. High quality solutions reproduce while low quality solutions do not reproduce. During reproduction, variations are introduced into the reproduced solutions.

The basic outline of an evolutionary computation algorithm is as follows:

```
Generate a population of structures
Repeat
  Evaluate the quality of the structures
  Decide whether to terminate
  Select structures to reproduce
  Produce variations of the selected structures
  Replace some old structures with new ones
End repeat
```

There are many ways to implement the above steps. There are many choices of data structures. Structures can be generated randomly, or if some better structures are known, the initial population can be seeded with these. The quality of solutions is often measured by a numerical fitness, but sometimes solutions directly compete against each other. Variations can be introduced into individual structures by a mutation process, or pairs of structures can be combined in a crossover process. Structures can be replaced one at a time, or the whole population of structures can be replaced.

Evolutionary computation can be used for optimization—finding the best, or maybe just a high quality, solution to a problem. Evolutionary computation has several positive features as an optimization technique. It can be applied when the structure of the problem is not well understood, and it does not require that the objective function be smooth or continuous. (It is a very difficult research problem to characterize the optimization problems on which evolutionary computation works well.) Evolutionary computation algorithms can often be easily parallelized.

Evolutionary computation can be used to “program” the behavior of agents (including robots). Evolutionary computation will work best when there are not exact requirements for the behavior of the agent but rather the quality of the behavior can be evaluated by running a simulation. Solutions obtained by evolutionary computation will often tend to be robust, resilient, and adaptive.

2 Two examples

2.1 Finding baffles for a cook-stove

The text gives a good real-world example of the use of evolutionary computation to design a wood burning cook-stove.

2.2 A string evolver

This is a fairly trivial non-real-world problem used to illustrate evolutionary computation techniques. The representation is a string (or array) of printable ASCII characters. (According to Ashlock, there are 96 such characters with codes from 32 to 127.) In the example program, the string length is 20. The objective is to evolve strings to match a given reference string. The fitness of a string is the number of positions (loci) where it matches the reference string. The contribution of each string position (locus) to fitness is independent of the others (i. e., there is no epistasis), and thus this is an easy problem.

3 Models of evolution

The first “model of evolution” that Ashlock uses is combination of tournament selection, crossover, and mutation. The population is divided into groups of 4, and each group of 4 is sorted into increasing order of fitness. The first 2 individuals (less fit) will be replaced by offspring of the last 2 (more fit). Crossover is implemented as part of copying the first 2 individuals over the last 2 individuals. A crossover point is selected, and both of the last 2 strings are chopped into a left part and a right part, where the left parts of the 2 strings are of equal length. Then the left parts are copied differently than the right parts. (This will be explained in class, or you can read the code of `Experiment2.1.cpp`). Mutation is done by modifying exactly one character of each new string. This model is short and relatively simple to implement, but mixes up the selection, variation-producing, and replacement steps.

As part of Experiment 2.1, there are 6 additional methods of selection and replacement. Since many of you won't have textbooks, I will restate the definitions on page 37 of the text.

In **tournament selection**, you pick a group of n creatures and then pick the best. If a separate tournament selection is done to pick each parent, Ashlock calls this “double tournament selection”. As n increases, there is more selection pressure. In **roulette wheel selection** (also called roulette selection, fit proportionate selection, and proportional selection), creatures are picked in proportion to their fitness. In **ranking selection**, roulette wheel selection is applied to the ranks of the creatures with the least fit creature having rank 1. Roulette wheel selection can give very little selection pressure at the end of a run when the fitnesses of the population are grouped together. It can give more selection pressure than ranking selection when there is a strong fitness gradient. Roulette wheel selection is the standard selection model in population genetics—but then this may be used to define fitness.

In an easy problem like the string evolver, strong selection pressure is good because the population will not get stuck in a local optimum.

A “child insertion method” is also needed. The simplest method, called **generational** evolution, is to replace the whole population. This is a **non-elitist** method because the most fit creature may not survive from one generation to the next. At the other extreme, you may generate one or two individuals by selection, crossover, and mutation, and then insert them into the population by removing the same number of individuals. This method is called **steady state** evolution. In between, you can replace some fraction of the individuals of the population, and this is what Ashlock favors. There are several methods for picking the individuals to replace. In **random replacement**, random individuals are chosen. **Roulette replacement** and **ranking replacement** work the same

as the corresponding selection methods, except that inverse fitness is used. (Note the potential problem if a creature has zero fitness.) In **absolute fitness replacement**, the least fit members of the population are replaced. In **locally elite replacement**, the best two of two parents and two children replace the two parents. (This method combines a tournament selection method with a replacement method.) In **random elite replacement**, each newly generated creature is compared to a random individual of the population with the better creature surviving. In many of these methods, there is a question of how to handle ties, and in problems with large regions of flat fitness, this choice is can be very important.

4 ECJ

ECJ (Evolutionary Computation in Java) is a large object-oriented software package that is designed to provide the practitioner or researcher with many/most of the facilities for doing evolutionary computation. Most aspects of a run are controlled by parameter files.

Installing ECJ is a matter of downloading and decompressing/dearchiving the files, and then setting the environment variables `PATH` and `CLASSPATH` appropriately for your OS and the location of Java and the ECJ files. The `make` (or `gmake` in Solaris) can be used to compile all files in Unix/Linux systems.

In order to solve a problem that fits into the framework provided by ECJ, one need to provide two files, a parameter file (with extension `.params`) and a Java program file which defines the fitness of creatures. For tutorial 1, both of these files are provided in the directory `ecj/ec/app/tutorial1`.

In tutorial 1, the problem solved is the Max Ones problem, namely to find a binary string with the maximum number of 1's. This is the binary version of the string evolver problem.

To run the program, you do the command:

```
java ec.Evolve -file tutorial1.params
```

Note that the name and location of the problem files is specified in the parameter files.

There will be output both to the terminal and to a file `out.stat` in the directory where you are when you run the command.

More explanation is given in the tutorial documentation pages.