

CS 495/595: Evolutionary Computation
Lecture notes for January 25, 2008
Ashlock's Software and ECJ, Selection and Variation Methods
Alden H. Wright, Department of Computer Science
University of Montana, Missoula, MT 59812
alden.wright@umontana.edu

Reading: Chapter 2 of the Ashlock text.

1 Comments on the software

Today I am passing out accounts on `kerlee.cs.umt.edu`. This is a fast 4-processor Linux computer. It has the easy-to-use but not-so-powerful command-line text editor `nano` installed. In this text editor, you use the arrow keys to move around in the file, and there is no separate insert mode like `vi/vim`. The special commands are control-key sequences which are shown at the bottom of the window. This is a good text editor to get started with. If you will use Unix/Linux a lot, I do suggest that you learn either `vi/vim` or `emacs`.

1.1 Ashlock's software

Ashlock's software is set up to be compiled using the GNU `g++` compiler. It could probably be made to compile in Microsoft Visual Studio by changing some header files and system calls, but I am not going to give you any help in doing so. I will give you some help in using a Unix or Linux environment where `g++` is well supported. One alternative on a Windows computer is Cygwin which installs a Linux-like environment on top of Windows.

1.2 ECJ

ECJ should run on any system with the Java SDK installed, but some utilities (such as the "make" command) will work only on Unix/Linux. There are instructions on the ECJ website for making ECJ work with Eclipse. I would encourage you to do this.

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`.

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.

How do we make sense of which selection method is better? In general, if the problem is “easy” without local optima, strong selection and elitism is good. But if the problem is hard, then it may be important to maintain higher diversity in the population, including less fit individuals, and so weaker selection and less elitism may work better.

4 Representation

The simplest representation (data structure) for EC creatures is an array (or vector) of bits, integers, or floating point numbers. For the time being, these are the only representations that we will consider. If the representation is an array of integers or floating point numbers, then one usually wants to have upper and lower bounds on the array components. For example, Ashlock’s string evolver uses an array of integers for its representation, and these integers must be in the range from 32 to 127.

Biological terminology is often used for array based representations since a DNA or RNA genome is essentially an array-based representation. One array component is called a **locus**, and the value at that array component is called an **allele**. For example, if the representation is a binary string of length 100, then there are 100 loci, and each locus can have two possible alleles, namely 0 or 1.

5 Variation producing methods

Variation producing methods modify the data structures that represent creatures in an EC algorithm. The data structure that represents a creature is often called its **genome**.

A mutation method applied to a parent creature produces a single child creature. Usually, mutation methods are biased to making small changes to the genome. For example, with a bit string representation, the most commonly used mutation method is to flip each bit independently with a small probability. This means that there is both a probability of no modification to the genome, and also a very small probability of large changes to the genome. In the Experiment 2.1 program, Ashlock uses a different mutation method, namely changing exactly one array component to another randomly chosen value. For the string evolver problem, this is fine, but for harder problems, this mutation method may not be able to escape from local optima.

A crossover method is analogous to sexual reproduction. Standard crossover takes two parents and produces either one or two children. To do one-point crossover on an array representation, you choose a crossover point between two loci. This crossover point divides the genome of each parent into a left part and a right part. The child or children are produced by swapping these parts between the parents. For example:

```

Parent 1:  0 1 0 1 1 | 0 1 1 1 0 1
Parent 2:  1 0 0 0 1 | 1 0 1 0 1 1
Child 1:   0 1 0 1 1 1 0 1 0 1 1
Child 2:   1 0 0 0 1 0 1 1 1 0 1
crossover pt          ^

```

For two-point crossover, you think of the genome as being circular. Then two crossover points are chosen, and this divides the genome into two parts, and the parts of the parents are swapped. For uniform crossover, for each bit of the child a random choice is made as to whether it will come from the first or second parent.

Crossover is called **pure** (or **conservative** by Ashlock) if crossover between two identical parents yields children that are identical to the parents. Crossover is called **respectful** if the genome components of the children must agree with the parents in the components where the parents agree. One-point, two-point, and uniform crossover are all pure and respectful. But some widely used crossovers for other representations are not pure and respectful.

This is important because if the whole population converges to a single allele at some locus, then neither selection nor respectful crossover can produce any alternative alleles at the locus, while mutation can produce alternative alleles.