

Genetic Algorithms, Historical Markers, and Finite State Automata
Complex Adaptive Systems
CS 591, Section 7
Assignment 1
February 19, 2003

1 Introduction

The basic idea of this assignment is to see how the genetic algorithm works on a language-induction problem. In language induction, the learning system is presented with a set of symbol strings defined over a fixed-size alphabet. These strings, known as sentences, are exemplars of a language (any set of legal sentences defines a language). The induction problem is to figure out a minimal procedure for recognizing legal strings in the language (that is, the set of exemplar sentences). In our assignment we will be using finite-state automata (FSA) to represent the procedure.

A trivial solution to this problem would be simply to define the language of all possible strings over the alphabet. Then, any possible set of exemplars would be recognized by our FSA. To make the problem more interesting, we will supply exemplars of strings that are in the language (positive examples) and strings that are not in the language (negative examples). Your GA should discover FSAs that recognize all the positive examples and reject all the negative examples.

We will study two methods for representing FSAs using genetic algorithms: the fixed-size table method and the variable-length genome method with historical markers. We will suggest the basic representation strategy and you are welcome to use existing software, but you are expected to design and implement your own fitness function.

2 Assignment

You are free to choose any publicly available genetic algorithm software or to write your own using the language of your choice. One possible choice is a very simple genetic algorithm written in C by Gary Flake (see URLs below) which generates FSAs for playing the Prisoner's Dilemma.

1. **Due: March 5** Modify an existing GA (or write your own) to learn FSAs (language acceptors) using a simple rule-table representation. Test the GA on the sample data sets.
2. **Due: March 12** Modify your GA to incorporate variable-length genomes and historical markers, as described in the Stanley and Mikkulainen papers. It is recommended that you also use their idea of initializing the population with a homogeneous set of minimal machines (we will discuss in class what the minimal machine looks like). At least initially, it is recommended that you skip the other mechanisms such as fitness-sharing.
3. **Due: March 12** Apply the extended GA to the FSA examples from Parts 1 and 2 and compare the performance of the two methods.

3 Details

3.1 Finite State Machines

If you have never heard of a finite-state automata, you can find a description and formal definition in any automata theory book and in most compiler books (among other applications, finite-state machines are used for the lexical analysis phase of compilers). Here are some materials we found on the web:

1. <http://www.math.iastate.edu/danwell/ma378/chapter6.pdf>
2. <http://www.dcs.ed.ac.uk/teaching/cs1/CS1/Ah/Notes/FiniteStateMachines1.pdf>

If there is sufficient interest, we can hold an extra discussion session to review the basics of FSAs.

3.2 Software

Gary Flake's book *The Computational Beauty of Nature*, published by MIT Press, has an excellent collection of software that accompanies it. The software, including the genetic algorithm, is available from <http://mitpress.mit.edu/books/FLAOH/cbnhtml/source.html>. The general download page is: <http://mitpress.mit.edu/books/FLAOH/cbnhtml/download.html>. There are UNIX, Mac, and Windows versions, both binary and source. The basic, architecture-independent (i.e. UNIX or Windows) source archive is: <http://mitpress.mit.edu/books/FLAOH/cbnhtml/cbn-noarch-src+docs.tgz>. The most relevant program for us is called *gaipd* (GA for Iterated Prisoner's Dilemma).

You may also wish to examine or use the NEAT software described in our readings. That is available in several versions from: <http://www.cs.utexas.edu/users/nn/pages/software/software.html>.

Once you have your GA turning over and generating FSAs, you will find it convenient to print the FSAs out in graphical form. The program GraphViz, available from <http://www.research.att.com/sw/tools/graphviz/>, allows you to display FSAs graphically. Note: You will need to convert the format of your GA representation in to a format that Dot understands.

3.3 Datasets

Inputs will first list the legal letters in the alphabet (each letter separated by a single “ ”), followed by a blank line. Then a set of positive examples will appear, one per line. The end of the positive examples will be denoted by a blank line. Next, will be a set of negative examples.

For example, if the language we are recognizing consists of any string made up of an even number of occurrences of the letter “a,” the input file might look as follows:

```
a

aa
aaaa
aaaaaa

a
aaa
aaaaa
```

Each language example will occur in a separate file. An important distinction in supervised machine learning is that between training sets and test sets. Using a separate set of testing data from that used to train the system allows us to see how well the system generalizes from a small set of examples to a general concept. In some of our data sets, we will include a third set of examples, the test set, consisting of positive and negative examples.

3.4 What to hand in

You are allowed to work with one other partner, if you choose. On March 5 you are expected to hand in a working GA (source code listing), some plots of its performance on the text examples (e.g., mean fitness plotted against generation), printouts of example high-fitness FSAs, and a 3-5 page writeup. On March 12, you should hand in an augmented writeup that describes your new representation, any changes to the fitness function, your new results, the comparison between the two methods, and your conclusions.

The writeup should describe at least the following:

- The problem you are trying to solve (in your own words);
- How you set out to solve the problem, e.g., which genetic algorithm software you used, its basic algorithms (what kind of selection, crossover, mutation, etc.);
- The representation (how did you represent FSAs to the GA);
- The fitness function(s) you tried;

- The basic parameters for your runs (population size, generation time, crossover rate, mutation rate, etc);
- Inputs and outputs;
- Experimental design (what experiments did you run and why);
- Experimental results;
- Discussion and Conclusions (how well did your program work and why do you think it performed the way it did)?