Fall 2011 COSC 3P71 Artificial Intelligence: Assignment 2

Due: Wednesday, Oct. 29th, by 5:00 pm

Goal: Genetic algorithm implementation and application.

Languages: Any programming language compilable/runnable in labs/sandcastle.

Hand in: (i) A listing of your source code, and a listing of your program execution for representative sample runs, including the requested report. Include department cover page. (ii) Electronic submission of all your code and data. Use “submit3p71” to submit your code. Run this at the top-level folder of your assignment directory. Be sure to document how to run your program! The marker will use this to try your solution.

Task: Implement a genetic algorithm system for a Self-Avoiding Walk

The overall procedure for a simple GA will be something like this:

Procedure simple GA
begin
Read problem instance data;
set GA parameters;
generate randomly an initial population POP of size Pop_Size;
for gen = 1 to MAXGEN do
    evaluate fitness of the individuals of POP;
    select new population using some selection strategy;
    apply genetic operators, crossover and mutation;
endfor;
end;

The GA’s main modules are described as:

(a) Initial Population initializer: creates a population of size Pop_Size of randomized individuals as described in class.

(b) Selection: use Tournament Selection (you can add Roulette as an additional option if desired).
   Include Elitism (single best individual)

(c) Crossover: given two individuals, this creates two offspring. Implement your GA using the following crossover strategies independently:
   (i) N-Point crossover (try runs with n=1, n=2, and some other n of your choice)
   (ii) Possibly an additional crossover (see below)

(d) Mutator: given an individual, this creates a mutated individual. Implement your GA using the following mutation operators:
   (i) Single-locus-flip (randomly choose a gene, and give it a new direction)
   (ii) Block exchange (select two subsequences, and swap them)
   (iii) Possibly an additional mutator (see below)

(e) Fitness evaluation function: Total area covered before running out of steps or falling off the field

(f) Genetic algorithm system: This is the implementation of the GA system

(g) User parameters: population size, maximum generation span, probability of [crossover, mutation etc], tournament size (k).
Your GA program should permit the user to *easily* define his/her own genetic parameters (e.g., crossover rate, mutation rate, population size, maximum generation span, etc.). The program should allow testing for various problems/sizes, i.e., different field dimensions (width and height as separate values, as well as starting coordinates).

**Experimental Analysis**

Run your GA to compare the performance of the crossover and mutation operators mentioned above by using the following parameters *(and include elitism in all cases):*

(a) Crossover rate = 100%, mutation = 0%
(b) Crossover rate = 100%, mutation = 10%
(c) Crossover rate = 80%, mutation 0%
(d) Crossover rate = 80%, mutation 10%
(e) Establish your own parameter settings

For elitism: replicate the chromosome with the highest fitness value

Incorporate into your experiments your own innovative idea; For full marks, include an additional crossover of your own choosing/design, an additional mutation of your own choosing/design, or an additional representation. For whichever you pick, compare it against the best results you got for the rest of the experiments.

For each experiment mentioned above, run your GA at least 5 times. Use problem instance #2 below. Additionally, after you've found a setup that you think works well, use those settings for the other *required* instances.

Output the following to a file or standard output:

All GA parameters, including random number seed.

Per each generation: best fitness value, average population fitness value

Per each run: best solution fitness and its corresponding best solution chromosome

Finally compute for the multiple runs: average of best fitness per generation and average population fitness per generation. Using a graph drawing tool such as excel, plot well labeled graphs for experiments above. Types of graphs you plot for the bonus part depend on your incorporated idea, if any. Feel free to experiment with different crossover and mutation rates. You will set your own Pop-Size and generation size.

Lastly, prepare a summarized typed report with sub-headings (a) Objective and problem definition b) Summary parameters used c) Results: that is, summary tables and graphs, EXPLAIN your graphs in detail. (d) Discussions and conclusions from your results. Your discussions should include issues like which crossover performed better than the other one, which mutation performed better, which representation was better, etc. If you included local search, did it help? How did the choice of GA parameters affect the final outcome etc? Make sure to include at least one rendered (visual) solution for each problem instance.

IMPORTANT NOTE: Pay attention to details and provide everything that I have asked for. The main aim of this assignment is to evaluate if you understood the basics of a genetic algorithm, hence a report without a working GA is problematic. On the other hand, a correct code, without proper DETAILED EXPERIMENTAL ANALYSIS and REPORT will not earn top marks.
**Self-Avoiding Walk:**

When presented with a grid (or other environment), a self-avoiding walk is a path that is meant to cover as much of that grid as possible. Since it has a limited distance it can travel, it should try to avoid repeating any areas it has already traversed (thus, self-avoiding).

A problem instance consists of a W×H grid and some starting coordinate.

The maximum distance the agent can travel is equal to the number of traversable squares on the grid. The fitness function is the number of unique (traversable) squares traveled by the agent. As such, the maximum path length is also the maximum fitness value for a given problem instance.

**Important:** If the agent doubles back onto an existing path, it simply doesn't earn any extra fitness. It can still proceed. However, if it drops off the grid (e.g. tries to travel to 3,-1), then that transcription stops immediately (i.e. it stops earning fitness).

Example – Consider the following path:

```
\[\text{S} \]
```

It scores a point for its starting square. It scores two more points when it travels north. The east and south bring the total up to 5. When it travels west, it doesn't earn a point for the first square (since it's already been visited), does earn a point for the next square, and then falls off the grid (ignoring whatever the rest of the path might have been), yielding a final fitness value of 6.

**Required Problem Instances:**

```
  
  
  S
  
```

```
  
  
```

```
  
  
```

```
  
  
```

```
  
  
```

```
  
  
```

```
  
  
```

```
  
  
```

```
  
  
```

```
  
  S
```
Bonus Problem Instances:

Include all of these problems (including statistics of your GA's performance on them) for a small bonus. Feel free to include any theories about how it tries to solve them.