Assignment 2

CS 482/682: Artificial Intelligence Fall 2013 Max Score: 100

Objectives

1. Learn and demonstrate knowledge of problem solving as local search
2. Learn and demonstrate knowledge of modeling two-person games as adversarial search

1 Local search

Use a hill climber and a genetic algorithm to maximize the function `double eval(int *vec)` contained in the three object files on the main class page. These are three **black box** functions and I want you to use the same hill climber and genetic algorithm with exactly the same parameters to find the maxima. **Black box** means that you do not know what the function does – all you know is that if you send the function a `int *vec`, it will return a `double` ≥ 0 – hence the use of `.o` files. You must use C or C++ for this part.

Here's a sample `main` that shows how to call `eval` in C++.

```
#include <iostream>
using namespace std;
double eval(int *pj);
int main()
{
  int vec[150];
  int i;
  for(i = 0; i < 150; i++){
    vec[i] = 1;
  }
  cout << eval(vec) << endl;
}
```

You can use the GA code from the Evolutionary Computing Systems Lab (ECSL) research group (linked on the main class page), the code from the book's website, from elsewhere, or write your own.

If you use ECSL code, here are the step by step instructions to modify the canonical GA to run with our evalaution functions.

1. Download the ga code and save it an directory called as2
2. `cd as2; tar xvzf ga.tgz` should create a directory called `ga` with all the code in the directory
3. `cd ga; make` should compile and link all the files and create an executable
4. `./ga infile < rx` should run the GA on the $f(x) = x^2$ sample function with the random seed in `rx` used to seed the random number generator and print running statistics on the terminal.
5. Replace `eval.o` with one of the `eval.o`s given to you
6. Edit `gen.c` and `init.c` to replace references to `eval`. That is:
   (a) Change the function prototype declarations to `double eval(int *)`
   (b) Change the call to `eval` to `eval(pi->chrom)`
7. Change the chromosome length in the input file: `infile`. This is the number on the second line and should be 150.
8. `./ga infile < rx` should now run the GA on the black box evaluation function

Assume that `vec` can only contain 0's or 1's, that is, it is a bit string. Going thru all 2^{150} combinations of the 150-bit vector is not a viable option

You will need to experiment with genetic algorithm parameters including population size, selection strategy, crossover types and probability, and mutation. Your genetic algorithm should not fail. It should reliably (more than 50% of the time) find the correct solution. If you have trouble getting one of the ECSLent GAs at the link above to work with the black box functions above, let me know as soon as you run into trouble.

You will have to write your own hillclimber but this need only be a few lines. Do not spend more than a couple of hours writing the hill-climber. You may spend a lot more time tweaking it if you like.

2 Turning in part 1 of your assignment

1. Turn this in to our grader at your appointed time
   (a) Your FULL name(s) and email address(es)
   (b) Write and turn in a typeset report. Describe all the differences from the canonical genetic algorithm and all parameter values. Show the graphs of avg-max, avg-avg, and avg-min fitness (y-axis) versus number of evaluations (x-axis). You will run your algorithms at least $n = 10$ times with different random seeds since you will need to show that your GA finds the correct solution more than 50% of the time (in this case > 5 times out of $n = 10$).

(c) To make things clearer. I expect six (6) graphs from you. Averages are over $n = 10+$ runs

   i. Hillclimber running on f0 (Number of evals versus avg-fitness)
   ii. Hillclimber running on f1 (Number of evals versus avg-fitness) this should look exactly the same as
   iii. Hillclimber running on f11 (Number of evals versus avg-fitness) which should take much much longer to run
   iv. GA on f0 (Generations versus avg-max, avg-avg, avg-min)
   v. GA on f1 (Generations versus avg-max, avg-avg, avg-min). Again these graphs should be exactly the same as
   vi. GA on f11 (Generations versus avg-max, avg-avg, avg-min) which should take much much longer to run.

(d) I also expect you to tell me on your report: 1) Your GA's reliability, the number of times / n, you exceeded 50 in fitness. 2) What was the average number (over n) of evaluations need to reach 50.

2. **Note: I only want source code for your hill-climber**

3. Write a short report the following:

   - For the hill climber, please describe the algorithm in words, and in pseudocode using the pseudocode style used in our textbook.
   - Describe a search space in which your hill-climber will fail (fail with high probability if it is non-deterministic)
   - For the genetic algorithm, please provide the population size, the number of iterations, the probability of crossover and mutation. If you changed selection, describe your changes. If you changed any of the other operators, describe these changes.

3 Adversarial Search

Write a program that plays Connect-4 well. Your program should not lose when given a large enough minimax tree depth.

Specification

Use minimax search with $\alpha - \beta$ pruning and user selectable depth to speed up your program. The program should allow the user the choice of specifying who makes the first move and it should print out the board after each move. For this game your program should not lose (ever) above a certain search depth. The program should also do error checking so that a user cannot cheat.

Your program should user the following optional command line arguments

- User selectable depth (-d depth)
- Whether or not to use $\alpha - \beta$ pruning (-A means to use $\alpha - \beta$ pruning)
- For graduate students only: N in Connect-N (-n N), board size in terms of the number of columns (-c nColumns) and rows (-r Rows).

Turning in Part 2

You need to do two things:

1. Turn in a hardcopy of source code and a script (or link to a movie) of your interaction with the program that shows off its strengths and weaknesses. In addition, we need a writeup of your assignment that includes,
   (a) A typeset explanation of your program's design: logic, data structures and control. Justify your static evaluator.
   (b) A typeset explanation of how to use your program (A man page).

   Finally, place the program executable on your web page; tell me where it is (URL) in your handin. Test and make sure you can download the executable from your web-page and can run the executable after downloading.
2. Sign up for a demo on the sheet handed out in class or send email to our grader.

Graduate Students

–

Ask me (sushil@cse.unr.edu) if you have questions.