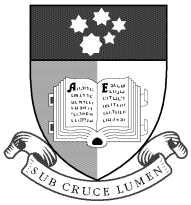


Department of Computer Science
University of Adelaide

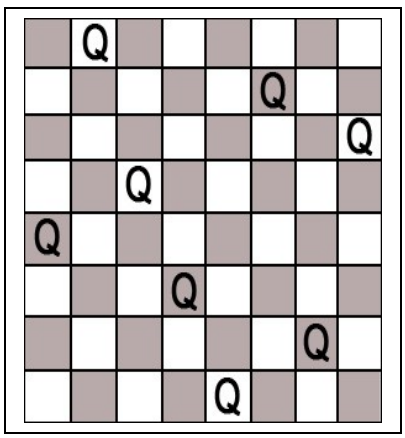
Artificial Intelligence

Assignment: Evolutionary Algorithm for the N by N Queen Problem

Outline

In this practical you are asked to develop an evolutionary system. This system may take as input a configuration file, specifying several parameters describing the *population size*, *number of generations*, as well as *selection type*, *type of crossover*, *type of mutation*, and the probabilities specifying *crossover* and *mutation rates*.

You should develop your system to solve the N by N Queens Problem. In the game of chess, the Queen can move horizontally, vertically or diagonally for as far as needed to capture pieces. The N by N Queens problem requires you to place one Queen on each row and column, in a way that none of the Queens can capture the other pieces. A typical 8 by 8 Queens Problem solution can be seen in the figure below.



So the N by N Queens problem involves creating a board of size n^2 and finding where to place the queens on the board without any collisions.

For more information on the N by N Queens problem, please look at:
http://www.math.utah.edu/~alfeld/queens/queens.html
or
http://en.wikipedia.org/wiki/Eight_queens_puzzle

Implementation

For this assignment you have been given skeleton code which you must complete. You are must not change the existing code or the input/output of the program, however you are free to add to the code as needed. The code can be found on the course website.

The code given implements the skeleton of a genetic algorithm used to solve the N by N Queen problem. The genetic algorithm uses a chromosome of size *n* (the number of board columns) where each gene in the chromosome represents the row position of the queen. For example, the board depicted above would have the following chromosome (with index 0-7): [4, 0, 3, 5, 7, 1, 6, 2].

The evolutionary process includes mutation and crossover operators, where parents must be selected from the population for 'breeding'. The variation and selection operators must be implemented in your code, with a basic implementation including single point crossover and simple mutation implemented with tournament or roulette wheel selection.

If you finish the basic implementation of the evolutionary process, then for further marks you might like to attempt different types of crossover, mutation and selection. The skeleton code has been designed to allow you to implement different types of mutation and crossover, and different selection mechanisms, and you should create different configuration for using these.

Testing

You should test your system on grids of different sizes. For example you should be able to solve grids of n=8, n=12 and n=18, and even harder ones with better algorithms implemented (with increased generations and population sizes for the harder solutions).

Solution

Your java solution should (at a minimum) consist of two separate programs, GeneticAlgorithm.java and NQueen.java. NQueen.java should be run with the command line

java NQueen <*n*> <seconds> <config file>

where <n> specifies the number of rows/columns of the board to run the solution on. The <seconds> field specifies how much time NQueen should spend on finding the solution. The <config file> specifies which configuration file to use.

For marking reasons, there should be no other output added to the program.

You must have at least one config file for this assignment. You may find some strategies (or different variation operators) perform better than others for different graph sizes, so you can suggest appropriate configs for boards of sizes 8, 18 and 24. You have also been given an example config file to use.

Submission

Submission is to be done using the svn repository (more information on svn can be found here: http://www.cs.adelaide.edu.au/docs/svn-instr.pdf).

Step 1: Creating the assignment directory in your svn repository

Open a terminal window on your machine, and use the following command (replacing your 'username' for your username):

svn mkdir -m"assignment intro" https://version-control.adelaide.edu.au/svn/username/ai-07-t1-prac1

This command will create an empty directory named ai-07-t1-prac1 in an svn repository that we have set up for you on the machine version-control.adelaide.edu.au. Your svn repository is called https://version-control.adelaide.edu.au/svn/username/ (with your username replaced).

Step 2: Checking out a working version of your assignment.

In your home account, change to a directory above the one where you want to place your working files for this assignment. Then type (replacing the username):

svn checkout https://version-control.adelaide.edu.au/svn/username/ai-07-t1-prac1

A new directory called ai-07-t1-prac1 will now be created in your current working directory. The contents of this now become a working-copy of the files for this assignment. Note that this working copy will look the same as your original copy but, the directory contains hidden sub-directories needed by svn to operate correctly.

Note that you can have more than one working copy of a project. For example you could have one working copy at home and one at university and you can use svn to help keep them in sync. See the svn documentation for details on how this can be done. However, for now, we will assume you have just the one working copy.

Step 3: Download the starting files

Download the .zip file:
https://www.cs.adelaide.edu.au/users/third/ai/2007Adelaide/assignments/prac1.zip .

This contains the starting files in a directory named prac1.

Step 4: Adding the starting files

Copy the directory pt into the directory ai-07-t1-prac1 that you created in step 2. Change to the directory ai-07-t1-prac1 and cut-and-paste the following commands:

```
svn add *.java
svn commit -m"startup files"
```

You should get a few messages describing what the add and commit commands are doing. By the end of these commands a copy of the starting files for your project will now be part of your svn repository.

The assignment requires you to edit the skeleton code given, and create new classes (if required). For each new file added, you need to repeat step 4 with the new java file name given instead of *.java.

Step 5: Saving your files in your repository

As you work on your assignment, you should commit your changes early and often. That is, you should regularly copy new versions of your files to your repository. The version control system ensures that previous versions are not lost so you can retrieve files you accidentally delete or go back to an earlier version if you make a mistake.

To commit your files to the repository you change to the directory containing the working copy of your files and type:

```
svn commit -m"meaningful message about what I just changed."
```

note, that the message after the -m in the line above can be anything. However, it should be relevant to the state of the project at this moment. Also note that running commit is the only way your repository will get to know about the latest versions of your files.

Step 6: Assignment Submission

The next step is to submit your assignment using the Computer Science Web Submission System.

By now you have written your assignment, committed your changed files to the repository along the way and thoroughly tested them. The Web Submission System will automatically mark your assignment using the current versions of your files in your repository. When you attempt to make a new submission you will be presented with an assessment cover sheet. This includes a link to your repository so you can check what files you are submitting. The cover sheet also includes a declaration that you are submitting your own work.

Note that, if the testing highlights some errors then you can always edit your files, commit them and make a new submission using the Web Submission System.

Submit all files by 5pm**, 23rd April 2006**.

Marking and consulting

The marks for the assignment will be based on a number of factors:

- The quality of your code (Good structure, commenting, naming etc..)
- The efficiency of your solution (the speed and results of the program given the size of the board etc..)
- The different types of extra variation or selection operators used.
- Experimentation e.g. knowing that a certain combination of mutation and crossover methods works well on larger boards, or that smaller population sizes and generations are needed for easier boards etc..

Your system will be tested on a number of different sized boards (e.g. 8, 12, 18 and 24). Checks will be made on your implementation of the various modules of selection, crossover, etc, and the quality of your code. Please use the bulletin board if you have questions. Note that you should not seek a solution via the bulletin board, but rather should make general inquiries. Likewise, students should not reveal partial solutions or sensitive, assessable aspects via the bulletin board. Solutions will be checked for evidence of copying.

Information on tournament selection

The tournament selection (with tournament size = 2) is quite simple. You select 2 random individuals from the population (you do not pay attention to the fitness scores at this stage) and the better one (with larger fitness) is selected for the next generation. You repeat this process population_size number of times, i.e., until you get a new population.

Mike Brooks
Zbigniew Michalewicz
Rhys Hill
Phillipa Avery
March 2007