1 Genetic Programming Assignment

- References:
  - Book: Artificial Intelligence: A New Synthesis, Chapter 4: Machine Evolution
2 Genetic Programming

- Evolution is central to biology in general, and bioinformatics in particular.
- To show you that evolution can indeed work and produce interesting results, you will be asked to write a Haskell program to evolve the guidance code for a wall-following robot.
- The idea of evolving computer programs automatically (instead of coding them yourself) is a major area of AI called genetic programming.
3 Genetic Programming

- Genetic programming achieves this goal of automatic programming by evolving a population of computer programs using the principles of Darwinian natural selection and biologically inspired operations.

- The operations include:
  - reproduction
  - mutation
  - crossover
  - architecture-altering operations (patterned after gene duplication and gene deletion in nature)
4 Overview

Flowchart for Genetic Programming

- Gen = 0
- Create Initial Random Population
- Termination Criterion Satisfied? Yes → Designate Result No → Evaluate Fitness of Each Individual in Population
- individuals = 0
- Gen = Gen + 1
- individuals = M?
- reproduction
  - Select One Individual Based on Fitness
  - Perform Reproduction
  - Copy into New Population
  - individuals = individuals + 1
- mutation
  - Select One Individual Based on Fitness
  - Perform Mutation
  - individuals = individuals + 1
5  Fitness Function

- most difficult part of GP is the fitness function
- it varies greatly from one type of program to the next
6 Haskell Assignment

- major project
- task is to evolve control system for wall-following robot
- shows you that evolution can work but results are messy
- worth 25% of your mark
  - 15% basic implementation
  - 10% research component
- to be done in pairs
<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>(1,1)</td>
<td>(2,1)</td>
<td>(3,1)</td>
<td>(4,1)</td>
<td>(5,1)</td>
<td>(6,1)</td>
<td>(7,1)</td>
</tr>
<tr>
<td>X</td>
<td>(1,2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(8,2)</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>(1,3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(8,3)</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>(1,4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(8,4)</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>(1,5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(8,5)</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>(1,6)</td>
<td>(3,6)</td>
<td>(4,6)</td>
<td>(5,6)</td>
<td>(6,6)</td>
<td>(8,6)</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>(1,7)</td>
<td>(3,7)</td>
<td></td>
<td></td>
<td>(6,7)</td>
<td>(8,7)</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>(1,8)</td>
<td>(2,8)</td>
<td>(3,8)</td>
<td></td>
<td>(6,8)</td>
<td>(7,8)</td>
<td>(8,8)</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
8 Boolean Functions

- \( \text{AND}(x,y) = 1 \) iff \( x = 1 \) and \( y = 1 \)
- \( \text{OR}(x,y) = 1 \) iff \( x = 1 \) or \( y = 1 \) (or both)
- \( \text{NOT}(x) = 1 \) iff \( x = 0 \)
- \( \text{IF}(x,y,z) = y \) if \( x = 1 \); else \( z \) if \( x = 0 \)
9 Actions

- north: moves the robot one cell up in the cellular grid
- east: moves the robot one cell to the right
- south: moves the robot one cell down
- west: moves the robot one cell to the left
10  Evaluating Actions

- actions have effects but no values
- the evaluation of any of them terminates the program so that there is no need to pass a value up the tree
- all the actions have their indicated effects unless the robot attempts to move into the wall in which case the action has no effect except to terminate the program
11 Sensory Inputs

- nw, n, ne, e, se, s, sw, w
- equal to 1 iff cell not free

<table>
<thead>
<tr>
<th>nw</th>
<th>n</th>
<th>ne</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>R</td>
<td>e</td>
</tr>
<tr>
<td>sw</td>
<td>s</td>
<td>se</td>
</tr>
</tbody>
</table>
12 Wall-Following Program

(IF (AND (OR (n) (ne)) (NOT (e)))
  (east)
  (IF (AND (OR (e) (se)) (NOT (s)))
    (south)
    (IF (AND (OR (s) (sw)) (NOT (w)))
      (west)
      (north))))
13 Sample Execution

```
X X X X X X X X X X X
X X X X X X X X X X X
X X X X X X X X X X X
X X X X X X X X X X X
X X X X X X X X X X X
X X X X X X X X X X X
```

13 Sample Execution
14 Sample Execution

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>33</td>
<td>34</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>32</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>31</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>28</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>13</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>27</td>
<td>23</td>
<td>X</td>
<td>X</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>14</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>X</td>
<td>X</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
15 Random Programs

- You must make sure that every function node has the required number of children in any random trees you create in generation 0.

- Observe that the mutation and crossover operations preserve this property — assuming that the initial random trees have it.

- As long as each function node has the required number of children, all programs in all generations can be executed.

- Thus, we can always compute the fitness of any program.
16 Tricky Evaluation Issues

- Book chapter somewhat vague, so here are a few clarifications.
- Evaluation is done (recursively) left to right.
- As soon as any action function is encountered, the action is taken and the program terminates.
- Example:

  \[
  \text{(NOT (NOT (NOT (AND (IF (NOT (south))}
  \quad (se))}
  \quad (w)))}
  \quad (NOT (n))]]])
\]
17 Tricky Evaluation Issues

• Example:

\[
\text{(NOT (NOT (NOT (AND (IF (NOT (south))
\text{(se))
\text{(w)))
\text{(NOT (n))))))}
\]

• evaluation starts at the root node, NOT, and proceeds recursively for the argument(s), from left to right

• in this case, we end up at the AND, and we evaluate its first argument

• we encounter an IF, so we evaluate its first argument \((\text{NOT (south)})\)

• evaluating the first argument of \((\text{NOT (south)})\) yields the action south

• south is executed and the program terminates without a value being returned

• consequently, the expression above is equivalent to simply “south”.
18 Tricky Evaluation Issues

- Evaluating AND
  
  \[(\text{AND} \ (0) \ \text{(south))}\]

Even though there is an action, evaluation of the first argument yields a 0, so the result of the AND is 0 and evaluation of the second argument is not done (which would otherwise terminate the program with the south action)

- Evaluating OR
  
  \[(\text{OR} \ (1) \ \text{(south))}\]

Again, even though there is an action, evaluation of the first argument yields 1, so the result of the OR is 1 and evaluation of the second argument is not done (which would otherwise terminate the program with the south action)
19 Tricky Evaluation Issues

- Evaluating IF
  
  \[(IF \ (south) \ (expr1) \ (expr2))\]

Since the condition is evaluated first, the program terminates with the south action.

\[(IF \ (1) \ (ne) \ (south))\]

Since the condition evaluates to 1, the second expression is evaluated but not the third; thus, this expression evaluates to (ne) even though there is an action in the 3rd argument to the IF.

\[(IF \ (0) \ (south) \ (ne))\]

Since the condition evaluates to 0, the 3rd expression is evaluated but not the second; thus, this expression evaluates to (ne) even though there is an action in the 2nd argument to the IF.
20 Fitness Function

- run program 60 times on room given (we only consider 1 fixed room)
- count the number of cells next to the wall that are visited during these 60 steps (such a cell may be visited multiple times but it counts only once)
- there are 32 cells next to the wall, so
  - a program that never gets to the wall would have a count of 0
  - a perfect program would have a count of 32
- do 10 of these runs with the robot starting at random starting positions
- the fitness of the program is equal to the total count of next-to-wall cells over the 10 runs
  - highest possible value is 320
21 Evolution

• Generation 0
  – we start with a population of 5000 random programs
  – these initial programs constitute generation 0

• Generation \((i + 1)\) is constructed from generation \(i\) in the following way:
  – 500 programs (10%) from generation \(i\) are copied directly into generation \(i + 1\) (reproduction)
  – 4500 new child programs (90%) are put into generation \(i + 1\) (crossover)
  – in this assignment, mutation is not done — and usually when it is used, it is used sparingly (perhaps at a 1% rate)
Reproduction Details (500 programs)

- Individuals are chosen for copying by the following *tournament selection* process:
  - 7 programs are randomly selected (with replacement) from the population of 5000
  - the most fit of these 7 programs is chosen

- it is possible to select the same individual multiple times, in which case we would have multiple copies of the exact same individual in the next generation

- we do not necessarily pick the most fit 500 programs

- even programs with low fitness have some chance of being chosen

- (sometimes always picking the best programs in every generation prevents a better solution from arising in the future from other programs that are not so fit right now)
23 Crossover Details (4500)

- a mother and father are each chosen from generation $i$ by tournament selection, using the same parameters as in reproduction

- two children are produced by swapping a random subtree in the mother with a random subtree in the father

- it is possible (though very unlikely) that we pick the same program for both mother and father; this presents no problems for the crossover operation

- the child program may or may not have higher fitness than its parents

- motivation for possible effectiveness is that a main program and a subexpression of fit parents are incorporated into their child
24 Crossover Details (4500)

Crossover Operation with Different Parents

Parents

\[
\begin{array}{l}
(+ (- \sqrt{(+2a)} b) (+2a)) \\
(- (- \sqrt{(+3a)} b) (+2a))
\end{array}
\]

Children

\[
\begin{array}{l}
(+ (- \sqrt{(+3a)} b) (+2a)) \\
(- (- \sqrt{(+2a)} b) (+2a))
\end{array}
\]

\[
\frac{\sqrt{b^2 - 2\theta + \sqrt{\theta^2 - 4\theta}}}{2\theta} = \frac{-b + \sqrt{b^2 - 4ac}}{2a}
\]
25 Crossover Details (4500)

Crossover Operation with Identical Parents

Parents

Children
26 Most Fit Individual in Generation 10

(IF (IF (IF (se) (0) (ne))
  (OR (se) (east))
  (IF (OR (AND (e) (0))
    (sw))
    (OR (sw) (0))
    (AND (NOT (NOT (AND (s) (se))))
      (se))))

(IF (w)
  (OR (north)
    (NOT (NOT (s))))
  (west))

(NOT (NOT (NOT (AND (IF (NOT (south))
      (se)
      (w))
    (NOT (n)))))))
27 Our Genetic Code

- also a result of evolution
- messy and hard to understand
- some regions of DNA may do nothing
28 Errors in Handout

- In figure 4.5, page 66, the third line of code is missing a closing bracket.

- In figure 4.7, the robot does not start moving right once it reaches the top given the code in that figure.