CS4618: Artificial Intelligence I

Genetic Algorithms

Derek Bridge
School of Computer Science and Information Technology
University College Cork

Initialization

In [1]:

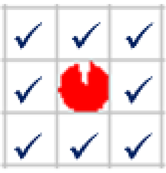
```
%reload_ext autoreload
%autoreload 2
%matplotlib inline
```

In [2]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Evolving Table-Driven Agents

- Previously, we programmed the agent: by filling in the table
- Today, we will **evolve** the agent
- To make the problem harder
  - …we will now allow all 8 sensors



  - …and 4 actions
    - MOVE
    - TURN(RIGHT, 2)
    - TURN(RIGHT, 4)
    - TURN(LEFT, 2)

Evolution by Natural Selection

- Successive generations differ from previous ones
  - Children inherit characteristics from their parents
  - But combining and mutating these characteristics introduces variations from generation to generation
- Less fit individuals are selectively eliminated (*'survival of the fittest'*)

Genetic Algorithms (GAs)

- GAs emulate evolution
- They iteratively update a **population** of **individuals**
- Each new generation of the population is obtained by probabilistically selecting fitter individuals from the current generation
  - Some of these individuals are **copied** to the next generation unchanged
  - Some are subject to **crossover** to create new offspring
  - Some of the new generation undergo **mutation**
- GAs differ from real-world evolution, e.g. each generation is the same size as the previous one

A genetic algorithm

- n is the number of individuals in the population
- χ is the proportion of the population to be replaced by crossover, e.g. 0.8
- μ is the mutation rate, e.g. 0.01

GA(n, χ, μ)

- // Initialise generation 0:
  $k = 0$;
  $P_k =$ a population of n randomly-generated individuals;
- // Evaluate P_k:
  Compute $fitness(i)$ for each $i \in P_k$;
- do {
  - // Create generation $k + 1$:
    // 1. Copy:
    Select $(1 - \chi) \times n$ members of P_k and insert into P_{k+1};
    // 2. Crossover:
    Select $\chi \times n$ members of P_k;
    pair them up;
    produce offspring;
    insert the offspring into P_{k+1};
    // 3. Mutate:
    Select $\mu \times n$ members of P_{k+1};
    invert a randomly-selected bit in each;
  - // Evaluate P_{k+1}:
    Compute $fitness(i)$ for each $i \in P_{k+1}$;
  - // Increment:
    $k = k + 1$;

  } while fitness of fittest individual in P_k is not high enough;
- return the fittest individual from P_k;

Representation of individuals

- Individuals are represented by *bit strings*
- This requires a way of **encoding** and **decoding**

Encoding/decoding

- Suppose this is the agent's table:

Percept	Action
00000000	MOVE
00000001	TURN(LEFT, 2)
00000010	TURN(RIGHT, 4)
⋮	⋮

- We can assign unique codes to the actions:

Action	Bt string
MOVE	00
TURN(RIGHT, 2)	01
TURN(RIGHT, 4)	10
TURN(LEFT, 2)	11

- We can concatenate all entries in the table to form one long bit string:

  0000000000 0000000111 0000001010 …
- Class exercise: How long will this bit string be?
- In fact, we don't need to include the percepts:

  00 11 10 …
- Class exercise: How long will this bit string be now?
- Class exercise: How many different bit strings (or tables or agents) are there?

Fitness

- The GA needs a (task-specific) **fitness function**
- E.g. place an individual into a room then, of all the cells that it visits, calculate the proportion that are adjacent to walls
- Typically, *average* performance over *several tasks* is computed

Copy

- How do we select the $(1 - \chi) \times n$ individuals who will be copied over?
- Obviously, influenced by their fitness, but we don't simply take the *fittest*
- Instead, it is probabilistic, e.g.:
    - **Roulette wheel selection**:
        - Probability of selection is equal to relative fitness:
$$Prob(choice = i) = \frac{fitness(i)}{\sum_{j=1}^{n} fitness(j)}$$
    - **Rank selection**:
        - Probability of selection is inversely proportional to position in the population after sorting by fitness
    - **Tournament selection**:
        - Repeatedly, select a random subset of the population and chose the fittest in this subset
- Selection is usually done *with replacement*: an individual can be picked more than once

Crossover

- In crossover:
    - Select $\chi \times n$ individuals
        - How? By roulette wheel, rank or tournament selection
    - Pair them up, giving $(\chi \times n)/2$ pairs
    - Swap a random portion of the father with a random portion of the mother, giving two new offspring
- The offspring may or may not be fitter than their parents:
    - We hope roulette wheel/rank/tournament seletion has chosen reasonably fit parents, and the offspring might have some fitness advantage by incorporating parts of these parents
    - On the other hand, no guarantees!
- There are different kinds of crossover:
    - **Single-point**: choose a random position
    - **Two-point**: choose two positions and swap the segment between them
    - **Uniform**: individual bits are chosen at random for swapping

Single-point crossover

11101001000 11101010101

00001010101 00001001000

Two-point crossover

11101001000 00101000101

00001010101 11001011000

Uniform crossover

11101001000 10001000100

00001010101 01101011001

Efficient single-point crossover

- Generate two masks, e.g.:

$$mask_1 : 11111100000$$
$$mask_2 : 00000011111$$

- Then

$$offspring_1 = (parent_1 \wedge mask_1) \vee (parent_2 \wedge mask_2)$$
$$offspring_2 = (parent_1 \wedge mask_2) \vee (parent_2 \wedge mask_1)$$

Example of efficient single-point crossover

$parent_1$:	<u>11101</u>001000	$parent_2$:	00001<u>010101</u>
$mask_1$:	11111000000	$mask_2$:	00000111111
∧:	<u>11101</u>000000	∧:	00000<u>010101</u>

∨: 11101010101

Mutate

- Select $\mu \times n$ individuals from the *new* generation
    - How? Random with *uniform* probability, not by fitness

- For each selected individual, a bit is chosen at random and this bit is inverted
- E.g.

11101<u>0</u>01000 11101<u>1</u>11000

Efficient mutation

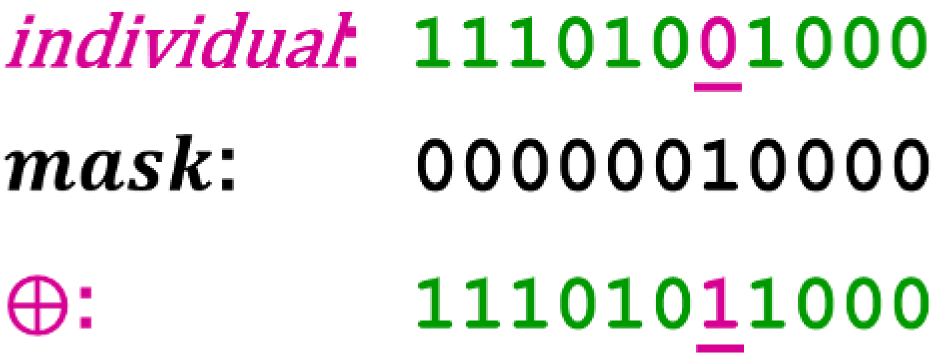
- Generate a mask, e.g.:

$$mask : 00000010000$$

- Then

$$newindividual = individual \oplus mask$$

  where $\oplus$ is exclusive-or
- E.g.



Discussion

- There's a risk of crowding:
  - This is where a fit individual reproduces a lot and it (or minor variants of it) dominate the population
  - It results in a lack of diversity and possible stagnation
- How to overcome overcrowding
  - Mutation
  - Rank selection or tournament selection
  - …

Applications

- GAs have been used to evolve:
  - Digital circuits
  - Factory schedules
  - University timetables
  - Neural network architectures
  - Similarity measures
  - …
- Lecture discussion: How would we do university timetabling using a GA? Are there difficulties?

```
In [ ]:
```