

**CS 441/541**  
**Artificial Intelligence**  
**Fall, 2008**

**Homework 6:**  
**Genetic Algorithms**

Due Monday Nov. 24.

In this assignment you will code and experiment with a genetic algorithm as a method for evolving control strategies for Robby the Robot.

**Robby, the Soda-Can-Collecting Robot**

Robby's job is to clean up his world by collecting the empty soda cans. Robby's world, illustrated in Figure 1, consists of 100 squares (sites) laid out in a  $10 \times 10$  grid. You can see Robby in site 0,0. Imagine that there is a wall around the boundary of the entire grid. Various sites have been littered with soda cans (but with no more than one can per site).

Robby isn't too intelligent, and his eyesight isn't that great. From wherever he currently is, he can see the contents of one adjacent site in the north, south, east, and west directions, as well as the contents of the site he's currently in. A site can be empty, contain a can, or be a wall. For example, in Figure 1, Robby, at site 0,0, sees that his current site is empty (i.e., contains no soda cans), the "sites" to the north and west are walls, the site to the south is empty, and the site to the east contains a can.

For each cleaning session, Robby can perform exactly 200 actions. Each action consists of one of the following seven choices: move to the north, move to the south, move to the east, move to the west, choose a random direction to move in, stay put, or bend down to pick up a can. Each action may generate a reward or a punishment. If Robby is in the same site as a can and picks it up, he gets a reward of 10 points. However, if he bends down to pick up a can in a site where there is no can, he is fined 1 point. If he crashes into a wall, he is fined 5 points and bounces back into the current

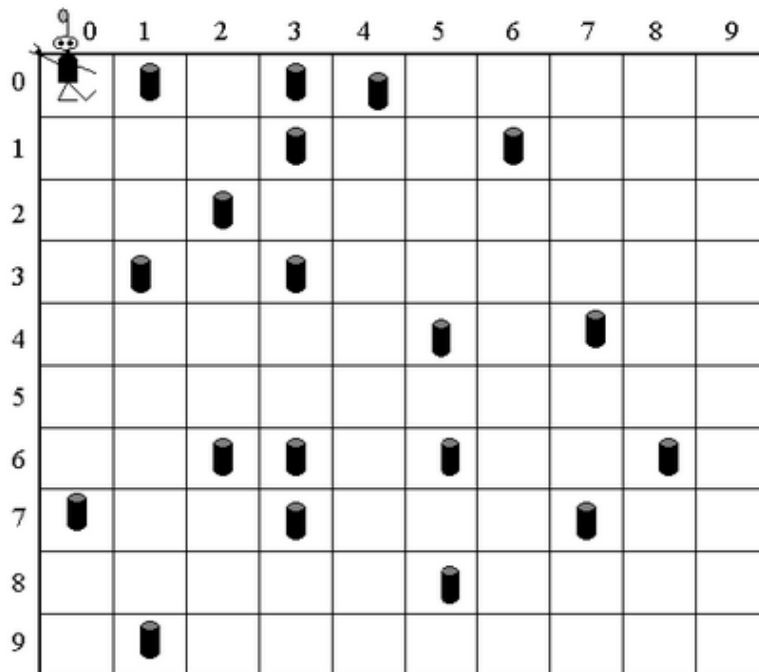


Figure 1: Robby's world: A  $10 \times 10$  array, strewn with soda cans.

site.

Clearly, Robby's reward is maximized when he picks up as many cans as possible, without crashing into any walls or bending down to pick up a can when no can is there.

Your assignment is write code for a genetic algorithm to evolve control strategies for Robby the Robot.

### Encoding of Strategies

Robby's strategy should be encoded as a look-up table that gives, for every possible *state*—i.e., situation he can encounter—the action he should take when in that state.

There are five different sites (north, south, east, west, current), each with

three possible types of contents (wall, empty, can). Thus there are  $3 \times 3 \times 3 \times 3 \times 3 = 243$  different “possible” situations. Of course this number includes some “impossible” situations, such as those in which Robby’s current site contain a wall (since Robby will always bounce back from a wall). However, you don’t have to filter these out; you can just leave them in the table, since they won’t have much affect on the GA.

Here’s an example of a strategy—actually, only part of a strategy, since an entire strategy would be too long to list here.

<b>Situation</b>					<b>Action</b>
<i>North</i>	<i>South</i>	<i>East</i>	<i>West</i>	<i>Current Site</i>	
Empty	Empty	Empty	Empty	Empty	MoveNorth
Empty	Empty	Empty	Empty	Can	MoveEast
Empty	Empty	Empty	Empty	Wall	MoveRandom
Empty	Empty	Empty	Can	Empty	PickUpCan
⋮	⋮	⋮	⋮	⋮	⋮
Wall	Empty	Can	Wall	Empty	MoveWest
⋮	⋮	⋮	⋮	⋮	⋮
Wall	Wall	Wall	Wall	Wall	StayPut

For example, Robby’s situation in Figure 1 is

<i>North</i>	<i>South</i>	<i>East</i>	<i>West</i>	<i>Current Site</i>
Wall	Empty	Can	Wall	Empty

To decide what to do next, Robby simply looks up this situation in his strategy table, and finds the corresponding action to take.

The “chromosome” to be evolved by the GA is just a listing of the 243 actions in the rightmost column of the strategy table, in the order given. Number the actions as:

- 0: move north
- 1: move south
- 2: move east
- 3: move west
- 4: move in a random direction
- 5: stay put
- 6: pick up can

Then the chromosome representing the strategy in the example above would be

0246 ... 3 ... 5

The GA remembers that the first action in the string (here action 0: MoveNorth) goes with the first situation (“Empty Empty Empty Empty Empty”), the second action (here action 2: MoveEast) goes with the second situation (“Empty Empty Empty Empty Can”), and so on. In other words, you don’t have to explicitly list the situations corresponding to these actions; instead the GA remembers the order in which they are listed.

Here are the steps in the genetic algorithm.

1. **Generate the initial population.** The GA starts with an initial population of POPULATION\_SIZE random individuals (strategies). As described above, each individual strategy is a list of 243 numbers, each gene between 0 and 6, which stands for an action (0 = *MoveNorth*, 1 = *MoveSouth*, 2 = *MoveEast*, 3 = *MoveWest*, 4 = *StayPut*, 5 = *PickUp*, and 6 = *RandomMove*). In the initial population, these numbers are filled in at random.

**Repeat the following for NUM\_GENERATIONS generations:**

1. **Calculate the fitness of each individual in the population.** The fitness of a strategy is determined by how well the strategy lets Robby do on NUM\_SESSIONS different cleaning sessions. A cleaning session consists of putting Robby at site 0, 0, and throwing down a bunch of cans at random (each site can contain at most one can; the probability

of a given site containing a can is 50%). Robby then follows the strategy for `NUM_ACTIONS_PER_SESSION` actions in each session. The score of the strategy in each session is the number of reward points Robby accumulates minus the total fines he incurs. The strategy's *fitness* is its average score over the `NUM_SESSIONS` different cleaning sessions (each of which has a different configuration of cans).

2. **Rank the population by fitness** Assign to each individual in the population an integer: 1 for the individual with highest fitness, 2 for the individual with next highest fitness, and so on. You can break any ties at random.
3. **Apply evolution** to the current population of strategies to create a new population. That is, repeat the following until the new population has `POPULATION_SIZE` individuals:
  - Choose two parent individuals from the current population probabilistically based on fitness rank. In more detail, the probability that each individual will be chosen is equal to:

$$\frac{\text{POPULATION\_SIZE} - \text{fitness\_rank} + 1}{1 + 2 + \dots + \text{POPULATION\_SIZE}}.$$

See the textbook, pp. 208-209, for a description of "roulette-wheel selection", which can be used here. In the book, roulette-wheel selection uses the actual fitness value of each individual; here we will use the *fitness\_rank* instead.

- Mate the two parents to create two children. That is, randomly choose a position in each number string; form one child by taking the numbers before that position from parent A and after that position from parent B, and vice versa to form the second child.
  - With a probability `MUTATION_PROBABILITY`, mutate numbers in each child. That is, for each number in the child's chromosome, with probability `MUTATION_PROBABILITY` replace that number with a randomly generated number between 0 and 6.
  - Put the two children in the new population.
4. Once the new population has `POPULATION_SIZE` individuals, return to step 2 with this new generation.

Finally, you can test the generalization performance of the highest-fitness strategy in the final generation by calculating its average score over 1000 new cleaning sessions (with `NUM_ACTIONS_PER_SESSION` actions in each session).

## Your assignment

Write code to implement the genetic algorithm, as described above. The algorithm should have the following default parameter settings:

<code>POPULATION_SIZE</code>	200
<code>NUM_GENERATIONS</code>	500
<code>NUM_SESSIONS</code>	200
<code>NUM_ACTIONS_PER_SESSION</code>	200
<code>MUTATION_PROBABILITY</code>	0.005

## Experiments:

Come up with your own hand-devised strategy for Robby, and calculate and record its generalization performance.

Then, for each of the following genetic algorithm experiments, calculate and record the generalization performance of the fittest individual in the final generation, averaged over 10 runs of the genetic algorithm. (If there are more than one “fittest” individual, choose one of them arbitrarily.)

1. Run your code with the default parameters.
2. Run your code without crossover. I.e., each parent has one “cloned” child, which is then subject to mutation.
3. Increase the population size to 500.
4. Decrease the population size to 100.
5. Design your own experiment by changing one or more parameter values, or by changing something about Robby’s task (e.g., the reward structure).

Finally, for each experiment, choose one of the 10 runs per experiment and plot the highest-fitness in the population as a function of generation number.

## **What to turn in**

Turn in (electronically, not hard copy!) all your code with instructions on how to run it.

Also turn in (preferably by e-mail) a write-up on the experiments above. Your write-up should describe your hand-designed strategy, describe and give the results of your experiments, comparing these results with the performance of your hand-designed strategy, give your plots, and give your hypotheses about the results (e.g., hypothesize as to why there is a difference between the results of the original GA run and the run without crossover, and so on).

Please don't hesitate to ask questions (to me or via the class mailing list) if you don't understand something in this assignment.