Phuc Le
Phuoc Le
Matthew Maksim
10/7/2022

Experimental part

1. For this experiment, we create a 8x8 map with height ranging from 1 - 5. The way we decide the height for each square is by flipping an unbiased coin, if its heads treat it as a 1, it is tails, flip the coin again and if it's heads treat it as a 2, if its tails flip the coin again and so on. If it flips tails and it passes 5, we continue flipping with the number set back to 1. We then pick a random location that agent will start at on the map. The agent can see vertically, horizontally, and diagonally, and will move in the direction of the height 1 higher than current. The goal is for the agent to find the local maximum. We will create four different algorithms to run this hill climbing experiment and they are local search, local search with 1 restart, local beam search, and simulated annealing algorithm.

For every experiment, we will show one trial in its entirety and only the results for the rest because the print is too long.

Hypothesis: For our experiment, we hypothesize that local search will have the lowest success rate, followed by local search with 1 restart, local beam search, and then with the highest success rate the simulated annealing algorithm. We think this is so because local search will have a high chance of getting stuck in a local maximum well, while local search with 1 restart and local beam search have 2 chances to find the maximum height, and simulated annealing algorithms have methods of escaping local maximum wells.

2. For this section, we created a hill climbing algorithm that will try to find the highest square it can reach. The agent can move sideways and can only climb to a square that is higher than the current square by 1. We ran 10 trials, with a different map and with the agent taking 20 steps each. The code for this experiment can be found in the file experiment2.py.

```
Trial 1:
[2, 5, 1, 1, 1, 1, 1, 1]
[1, 1, 2, 5, 1, 2, 1, 1]
[2, 2, 1, 3, 1, 1, 1, 1]
[2, 2, 3, 2, 1, 2, 5, 1]
[1, 1, 2, 1, 1, 1, 3, 2]
[2, 1, 2, 3, 1, 1, 2, 1]
[2, 1, 2, 1, 1, 1, 2, 2]
[3, 3, 1, 2, 2, 4, 1, 2]

Highest number: 5

starting at: ( 4 , 5 )
```

```
current height: 1

current position: (5, 6)
current height: 2

current position: (4, 6)
current height: 3

current position: (4, 6)
current height: 3

current position: (4, 6)
current height: 3

current position: (4, 6)
current height: 3

current position: (4, 6)
current height: 3

current position: (4, 6)
current height: 3

current position: (4, 6)
current height: 3

current position: (4, 6)
current height: 3

current position: (4, 6)
current height: 3

current position: (4, 6)
current height: 3

current position: (4, 6)
current height: 3

current position: (4, 6)
current height: 3

current position: (4, 6)
current height: 3
current position: (4, 6)
current height: 3

current position: (4, 6)
current height: 3

current position: (4, 6)
current height: 3

current position: (4, 6)
current height: 3

current position: (4, 6)
current height: 3

current position: (4, 6)
current height: 3
```

For the first trial, the agent got stuck in a local maximum, and could not find the global maximum. So for trial 1, the agent failed to find the global maximum.

Trial 2: pass
Trial 3: fail
Trial 4: fail
Trial 5: pass
Trial 6: fail
Trial 7: fail
Trial 8: fail
Trial 9: fail
Trial 10: fail

We repeated this experiment 9 more times for a total of 10 times. The total result we got was 2 passes and 8 fails, so from the experiment the rate at which the agent can find the global maximum is 20%.

3. For this section we try again with hill climbing but this time we will let the agent run for 10 steps and if it can't find the global maximum, we will restart the board and run it for 10 steps. The code for this experiment can be found in the file experiment3.py

Trial 1:

```
[2, 2, 1, 3, 1, 1, 3, 3]
[4, 3, 3, 2, 1, 5, 1, 1]
[2, 1, 1, 2, 1, 1, 2, 1]
[2, 1, 1, 1, 1, 2, 2, 3]
```

```
[2, 1, 2, 4, 1, 3, 2, 1]
[1, 5, 3, 1, 2, 5, 1, 2]
[1, 1, 1, 1, 1, 1, 2, 2]
[1, 4, 3, 2, 1, 3, 2, 1]

highest number: 5

starting at: ( 0 , 6 )
current height: 3

current position: (0, 7)
current height: 3

current position: (0, 7)
current height: 3

current position: (0, 7)
current height: 3

current position: (0, 7)
current height: 3

current position: (0, 7)
current height: 3

current position: (0, 7)
current height: 3

current position: (0, 7)
current height: 3

current position: (0, 7)
current height: 3

current position: (0, 7)
current height: 3

current position: (0, 7)
current height: 3

restart

current position: (7, 3)
current height: 2
current position: (7, 2)
current height: 3

current position: (7, 1)
current height: 4

current position: (7, 1)
current height: 4

current position: (7, 1)
current height: 4

current position: (7, 1)
current height: 4

current position: (7, 1)
current height: 4

current position: (7, 1)
current height: 4

current position: (7, 1)
current height: 4

current position: (7, 1)
current height: 4
```

For the first experiment, the agent got stuck in a local maximum after stepping 10 steps so it restarted and stepped another 10 steps. It again got stuck in a local maximum so this first trial was a failure.

Trial 2: fail
Trial 3: fail
Trial 4 pass
Trial 5: pass
Trial 6: fail
Trial 7: pass
Trial 8: fail
Trial 9: fail
Trial 10: fail

We repeated this experiment for another 9 times and the total result we got was 3 pass and 7 fail, so from the experiment the rate at which the agent can find the global maximum is 30%.

4. For this section, we find the highest point by using local beam search. There will be 2 beams for the search and the agent will take 20 steps. The code for this experiment can be found in the file experiment4.py.

Trial 1:

```
[5, 2, 2, 4, 4, 1, 2, 3]
[1, 1, 1, 1, 3, 1, 3, 2]
[2, 3, 2, 3, 1, 5, 4, 2]
[2, 3, 1, 1, 1, 2, 1, 1]
[2, 2, 2, 1, 2, 2, 3, 1]
[4, 2, 2, 5, 1, 1, 1, 1]
[1, 1, 1, 3, 1, 3, 2, 5]
[4, 2, 1, 1, 2, 3, 1, 2]

highest number: 5
starting at: ( 5 , 1 )
current height: 2
closest max height: (5, 3)

starting frontier: [(5, 2), (4, 2)]

frontier: [(6, 3), (4, 1)]
frontier: [(6, 3), (4, 1)]
frontier: [(6, 3), (4, 1)]
frontier: [(6, 3), (4, 1)]
frontier: [(6, 3), (4, 1)]
frontier: [(6, 3), (4, 1)]
frontier: [(6, 3), (4, 1)]
frontier: [(6, 3), (4, 1)]
frontier: [(6, 3), (4, 1)]
frontier: [(6, 3), (4, 1)]
frontier: [(6, 3), (4, 1)]
frontier: [(6, 3), (4, 1)]
frontier: [(6, 3), (4, 1)]
frontier: [(6, 3), (4, 1)]
frontier: [(6, 3), (4, 1)]
frontier: [(6, 3), (4, 1)]
frontier: [(6, 3), (4, 1)]
frontier: [(6, 3), (4, 1)]
frontier: [(6, 3), (4, 1)]
frontier: [(6, 3), (4, 1)]
```

For this experiment, both beams were unable to reach the global maximum. One beam got stuck at (6, 3) which is right next to the goal, but was unable to reach it because the height difference was more than one. The second beam was unable to reach the goal because it got stuck in a global maximum and was unable to escape. So this trial was a failure.

Trial 2: fail
Trial 3: fail
Trial 4: fail
Trial 5: fail
Trial 6: fail
Trial 7: fail
Trial 8: fail
Trial 9: fail
Trial 10: fail

We repeated this experiment 9 more times, and the result was that all runs failed to find the global maximum. This means that for this experiment the success rate is 0%.

5. For this section we did the simulated annealing algorithm. This is like normal hill climbing, but the agent can escape a local maximum by going downward. If the agent is at a local maximum with no higher step to take, it can step down a step with a probability. This makes sure that an agent will find the global maximum if given enough steps. The pseudo code in the slides was looking for a lower value so we had to edit it a little bit to make our agent find the highest value. The code for this experiment can be found in the file experiment5.py.

The edited pseudo code:

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
   inputs: problem, a problem
         schedule, a mapping from time "temperature"
   current := MAKE-NODE(problem.INITIAL_STATE)
   for t = 1 to infty do
      T := schedule(t) //typically as t gets larger schedule(t) is smaller
      if(T == 0) return current //when Temperature is 0 return state
      next := a randomly selected successor of current
      DeltaE := next.value - current.value //estimated change in energy
      if(DeltaE > 0) current := next
         else current := next with probability e^(DeltaE/T)
```

For our code we make T = schedule - t

Trial 1:
[3, 2, 3, 1, 1, 3, 1, 1]
[1, 3, 1, 2, 2, 2, 1, 1]
[1, 1, 3, 2, 2, 1, 2, 1]
[1, 2, 2, 1, 2, 1, 4, 1]

```
[3, 3, 1, 1, 1, 1, 2, 3]
[2, 3, 2, 1, 3, 3, 1, 2]
[1, 1, 2, 1, 1, 1, 1, 1]
[3, 1, 2, 2, 2, 2, 1, 1]

Max height: 4

starting at: ( 3 , 0 )
current height: 1

current position: (4, 1)
current height: 3

current position: (3, 2)
current height: 2

current position: (2, 1)
current height: 1

current position: (3, 0)
current height: 1

current position: (4, 1)
current height: 3

current position: (5, 2)
current height: 2

current position: (6, 3)
current height: 1

current position: (5, 4)
current height: 3

current position: (6, 5)
current height: 1

current position: (5, 6)
current height: 1

current position: (4, 7)
current height: 3

current position: (3, 6)
```

current height: 4

max reached

For trial 1, the agent was able to find the global maximum after stepping.

Trial 2: fail
Trial 3: fail
Trial 4: pass
Trial 5: pass
Trial 6: fail
Trial 7: pass
Trial 8: fail
Trial 9: fail
Trial 10: pass

We repeated this experiment another 9 times and in total it passed 4 times and failed 6 times. This means that the rate at which the agent passes is 40%.

Conclusion:

In conclusion, we found that normal hill climbing passes 20% of the time, hill climbing with restart passes 30% of the time, local beam search passes 0% of the time, and simulated annealing algorithms pass 40% of the time. This also matches up with our hypothesis, but there were some key differences. We were right that normal hill climbing would have the lowest probability of finding the maximum. Once the agent reaches a local maximum, it can't do anything. Hill climbing with 1 restart did a little bit better as when the agent reached a local maximum, the restart gave it another chance to try again. The local beam search was what surprised us the most. We think the reason the agent fails most of the time is because the way we determine the best score of each step. We used the euclidean distance to determine the score and since our agent could not step down to lower heights, our agent couldn't find a path to reach the maximum. The simulated annealing algorithm did the best as we had expected. The ability of being able to have a chance of escaping a local maximum increased the chance of finding the maximum.