CS440 Assignment2

Yeoun Chan Kim   John Strauser   Xuanang Wang

October 25, 2019

p1

Expansion sequence:
Start
(Lugoj, f=244, g=0, h=244)
Expand Lugoj:
(Mehadia, f=311, g=70, h=241)
(Timisoara, f=440, g=111, h=329)
Expand Mehadia
(Drobeta, f=387, g=145, h=242)
Expand Drobeta
(Craiova, f=425, g=265, h=160)
Expand Craiova
(Pitesti, f=503, g=403, h=100)
(Rimnicu Vilcea, f=604, g=411, h=193)
Expand Timisoara (440 ¡ 503 ¡ 604)
(Arad, f=565, g=229, h=336)
Expand Pitesti (503 ¡ 565 ¡ 604)
(Bucharest, f=504, g=504, h=0)
We can stop here because f(Bucharest) < f(Arad) < f(Rimnicu Vilcea)

p2

A)
BFS:
State 1 expands to State 2 and 3
State 2 expands to State 4 and 5
State 3 expands to State 6 and 7
State 4 expands to State 8 and 9
State 5 expands to State 10 and 11
State 6 expands to State 12 and 13
State 7 expands to State 14 and 15

State 8 expands to State 16 and 17
State 9 expands to State 18 and 19
State 10 expands to State 20 and 21
State 11 is found and the search stops

Depth-Limited:
State 1 expands to State 2 and 3
State 2 expands to state 4 and 5
State 4 expands to state 8 and 9
State 8 expands to state 16 and 17
Wont expand 16 and 17 due to depth limit of 3
State 9 expands to state 18 and 19
Wont expand 18 and 19 due to depth limit of 3
State 5 expands to state 10 and 11
State 10 expands to state 20 and 21
Wont expand 20 and 21 due to depth limit of 3
State 11 is found and the search stops

Iterative Deepening:
limit starts at 0
State 1 expands to state 2 and 3

limit is increased to 1
State 1 expands to state 2 and 3
State 2 expands to State 4 and 5
State 3 expands to State 6 and 7

limit is increased to 2
State 1 expands to state 2 and 3
State 2 expands to State 4 and 5
State 4 expands to State 8 and 9
State 5 expands to State 10 and 11
State 3 expands to State 6 and 7
State 6 expands to State 12 and 13
State 7 expands to State 14 and 15

limit is increased to 3
State 1 expands to state 2 and 3
State 2 expands to State 4 and 5
State 4 expands to State 8 and 9
State 8 expands to State 16 and 17
State 9 expands to State 18 and 19
State 5 expands to State 10 and 11
State 10 expands to State 20 and 21
State 11 is found and the search stops

B)
Bidirectional sort should be optimal for this problem since all edges have the same undefined cost.

Order:
State 1 expands to state 2 and 3
State 11 expands to state 5
State 2 expands to state 4 and 5
State 5 expands to state 2, which has already been expanded so the search is complete

Branching factor for the downward search (starting at state 1) is 2.
Branching factor for the upward search (starting at state 11) is 1.

p3

a) True
b) True
c) True
d) False
e) True
f) True
g) True
h) False
i) True

p4

Advantage: Iterative Deepening requires less memory overall compared to BFS.
Disadvantage: Becuase Iterative Deepening restarts the search each time the depth-limit is increased, the amount of nodes expanded is significantly higher.

part 8

1) A heuristic function h is said to be admissible if for any state n, the heuristic value of the state $h(n)$ is less or equal to the heuristic value of the goal state $h(g)$.
a) Given the fact that both $h1(n)$ and $h2(n)$ are admissible, $h(n)$ would be admissible because either $h1(n)$ or $h2(n)$ overestimate the cost to reach the $h(g)$.

In this case, the smaller one of those two would still obtain a cost which smaller or equal to the real cost to reach the h(g) therefore admissible.

b) We consider the given function at the boundary position:

$$\text{when w = 0, h(n) = h2(n)}$$
$$\text{when w = 1, h(n) = h1(n)}$$

Because of the fact that both h1(n) and h2(n) are admissible, we can see h(n) remains admissible for both boundaries.

Furthermore, we want to prove that the given function remains admissible for any value within the interval $0 < w < 1$. Again, because of the fact that both h1(n) and h2(n) are admissible, no matter which value in the interval is chosen, h(n) would always obtain a value which smaller than either h1(n) or h2(n) which would not exceed the cost to reach h(g) therefore is admissible.

c) Given the fact that both h1(n) and h2(n) are admissible, h(n) would be admissible because either h1(n) or h2(n) overestimate the cost to reach the h(g). In this case, even if we want to choose the larger one, it would still obtain a cost which smaller or equal to the real cost to reach the h(g) therefore admissible.

2) A heuristic function h is said to be consistent if for any state n, the heuristic value of the state h(n) is less or equal than the cost c(n') to go to state n' starting from n plus the heuristic value of that state h(n')

$$h(n) \le c(n') + h(n')$$

There are plenty of explanations being done in part 1), therefore we are going to explain 2) in a simple way. Through observation in part 1), we can notice that none of three functions would return a value which make h(n) larger than either of h1(n) or h2(n). Therefore, given the fact that both h1(n) and h2(n) satisfies the equation above, a value smaller or equal to h1(n) or h2(n) would still satisfies the equation above; therefore remain consistent.

Part 9

a) Hill climbing works better than simulated annealing when the objective function has only one global maximum and without plateaux.

b) The hill-climbing part of simulated annealing is not necessary when the maximum (peek) value in the objective function is relatively or completely flat.

c) Simulated annealing is useful when the local/global maximum of objective function is relatively sharp, the objective function has a lot local maximum with plateaux or shoulder situation.

d) The simulated annealing would return a value which is or near the global maximum. Because of the fact that it does not often return the best solution, given that we know the value (measure of goodness) of each state we visit, we
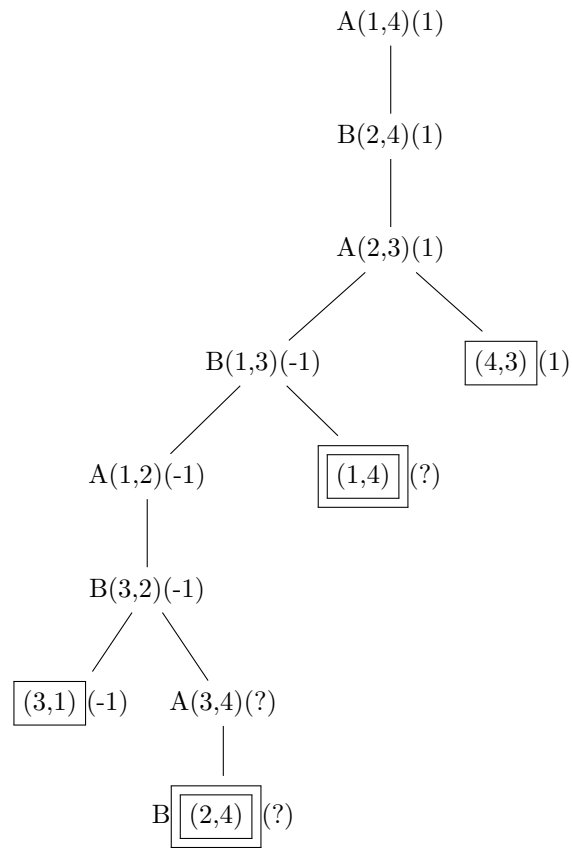
can compare the current state with it's neighbors and return the one which has the best measure of goodness.

e) Since the formal simulated annealing only stores the current and the next state, there would be a lot redundancy work being done when running the algorithm. However, given that we have enough memory to hold more information, we can simply start from the beginning of the objective function, traverse and store each state's measure of goodness until the end of the objective function. Then, we just pick the one which has the best measurement of goodness as return value.

f) The key of gradient ascent search is calculate the gradient of the objective function at current state and take step proportional to the gradient if it is positive. If the current state is at local maximum, all of it's neighbor would have a negative gradient and the agent would stuck.

Part 10

For the following tree, each state have a format of K(P1, P2)(C) where K represents which player's move, P1 represents the current token position for player A, P2 represents the current token position for player B, C represents the minmax value of the state. Note: the end state has no K.

```
                          A(1,4)(1)
                              |
                          B(2,4)(1)
                              |
                          A(2,3)(1)
                         /          \
                  B(1,3)(-1)      [(4,3)](1)
                   /       \
            A(1,2)(-1)    [[(1,4)]](?)
                |
            B(3,2)(-1)
              /    \
      [(3,1)](-1)  A(3,4)(?)
                       |
                  B[[(2,4)]](?)
```

We assume that both player wants to maximize his/her value while
minimizing his/her opponent's. Therefore, as player A's perspective, the game
would be ended at the line 4 therefore there is no need to worry about the "?"
values.