Vicky Prakash, Ananya Jana netID: vp407, aj611

Problem 1:
Tree search with BFS
Search queue
S(start state)
A, B
B, B, C, G, S
B, C, G, S, A, C, S
C, G, S, A, C, S, A, C, S
G, S, A, C, S, A, C, S
In next step, goal state G is reached.

Cost of this path = 2 + 2 + 1 + 3 = 8.

Tree search with DFS
Search queue
S(start state)
B, A
S, C, A, A
B, A, C, A, A
S, C, A, A, C, A, A
Infinite loop as S gets enqueued multiple times. This is because the graph is undirected.
As the search doesn't end, we dont have a path and associated cost.

Tree search with Uniform Cost
Search queue
S
A, B
B, C, G
C, G
G

In this case, same node is not enqueued twice, but its cost can be updated. **Search path**
S->A->B->C->G

Cost of this path = 2 + 2 + 1 + 3 = 8.

Tree search with Greedy best first
Search queue
S
A, B
A, C
A, G
Search path
S->B->C->G

Cost of this path = 6 + 1 + 3 = 10.

Tree search with A^*
Search queue
S
A, B
B, B, C, G, S
B, C, G, S, A, C, S
B, C, G, S, A, S, A, B, G
Search path
S->A->B->C->G

Cost of this path = 2 + 2 + 1 + 3 = 8.

Graph search with BFS
Search queue
S(start state)
A, B
B, C, G
C, G
G

Cost of this path = 2 + 8 = 10.

Graph search with DFS
Search queue
S(start state)
B, A
C, A

G, A

Cost of this path = 6 + 1 + 3 = 10.

Graph search with uniform cost
Search queue
S(start state)
A, B
B, C, G
C, G
G
Cost of this path = 2 + 8 = 10.

Graph search with Greedy best first
Search queue
S
A, B
A, C
A, G
Search path
S->B->C->G

Cost of this path = 6 + 1 + 3 = 10.

Graph search with A^*
Search queue
S
A, B
C, G, S, B
G, S, B
Search path
S->A->C->G

Cost of this path = 2 + 4 + 3 = 9.

Problem 2:
a) Admissibility check:
h_1 and h_2 are admissible, meaing they are an underestimate of the actual cost.

i) $h_{min}(n) = min\{h_1(n), h_2(n)\}$ So, $h_{min}(n)$ must be euqal to either $h_1(n)$ or $h_2(n)$. Hence $h_{min}(n)$ is an underestimate of the actual cost. This means h_{min} is an admissible heuristics.

ii) $h_{max}(n) = max\{h_1(n), h_2(n)\}$. So, $h_{max}(n)$ must be euqal to either $h_1(n)$ or $h_2(n)$. Hence $h_{max}(n)$ is an underestimate of the actual cost. This means h_{max} is an admissible heuristics.

iii) $h_{lin}(n) = wh_1(n)+(1-w)h_2(n)$. So, the value $h_{lin}(n)$ lies between $h_1(n)$ or $h_2(n)$. Hence $h_{lin}(n)$ is an underestimate of the actual cost. This means h_{lin} is an admissible heuristics.

a) Consistency check:
h_1 and h_2 are consistent, meaing they obey the triangle inequality rule.
$h_1(n) \leq c(n, n') + h_1(n')$ - (1)
$h_2(n) \leq c(n, n') + h_2(n')$ - (2)

i) $h_{min}(n) = min\{h_1(n), h_2(n)\}$ So, $h_{min}(n)$ must be euqal to either $h_1(n)$ or $h_2(n)$. Let's say $h_{min}(n) = h_1(n)$. Now, $h_1(n) \leq c(n, n') + h_1(n')$ from (1). And $h_1(n) \leq h_2(n) \leq c(n, n') + h_2(n')$. So $h_1(n) \leq c(n, n') + min\{h_1(n'), h_2(n')\}$ i.e. $h_{min}(n) \leq c(n, n') + min\{h_1(n'), h_2(n')\}$ or, $h_{min}(n) \leq c(n, n') + h_{min}(n')$. Hence, h_{min} is a consistent heuristics.

ii) $h_{max}(n) = max\{h_1(n), h_2(n)\}$. So, $h_{max}(n)$ must be euqal to either $h_1(n)$ or $h_2(n)$. Let's say $h_{max}(n) = h_1(n)$. Now, $h_1(n)c(n, n') + h_1(n')$ from (1). And $h_1(n') \leq h_{max}(n)$. So $h_1(n) \leq c(n, n') + max\{h_1(n'), h_2(n')\}$ i.e. $h_{max}(n) \leq c(n, n') + max\{h_1(n'), h_2(n')\}$or, $h_{max}(n) \leq c(n, n') + h_{max}(n')$. Hence, h_{min} is a consistent heuristics.

iii) $h_{lin}(n) = wh_1(n) + (1 - w)h_2(n)$
$\leq w * [c(n, n') + h_1(n')] + (1 - w) * [c(n, n') + h_2(n')]$
$\leq [(w + 1 - w) * c(n, n')] + [wh_1(n') + (1 - w)h_2(n')]$
$\leq c(n, n') + h_{lin}(n')$
Hence, h_{lin} is a consistent heuristics.

b) $w = 0$, $f(n) = 2g(n)$. This is uniform Cost search.
$w = 1$, $f(n) = g(n) + h(n)$. This is A^* search.
$w = 2$, $f(n) = 2h(n)$. This is Greedy Best First search.
We know that Uniform Cost search is optimal and A^* tree search with admissble heuristic is also optimal. So, the algorithm is optimal for $w = 0$ and $w = 1$.

c) In this case, we are allowed to pick a non optimal node in the A^* search. This may lead to an increase in the cost. However, if we use tree search, in future we may still pick the optimal node and enqueue it, whereas in graph search we will not enqueue the same node. So, with tree search we may still have an optimal solution at the end, but with graph search the cost is probably going to be higher.

Problem 3:
Input: Grid where some of the squares are blank and some are shaded. List of words from dictionary.
a) Formulation as a general search problem:
Start state: Empty grid
Goal state: Grid filled with meaningful words. In this case, there can be multiple possible number of grids with valid words
State space: Intermediate states i.e. when the grid is filled partially with words
Action: Pick word from dictionary and fit into a blank keeping the already entered letters in place
Transition model: As we fill words from the dictionary into the grid, we state of the grid changes.
Solution: The sequence in which the grid n be filled with valid words. There can be multiple such choices and sequences.
Since we can count the number of blanks, we have the number of words that will fit into the grid as well. We can pick words and start filling in blanks,preserving the already present letters. We can proceed in this way until we complete the puzzle or a contradiction is reached.

It is better to fill in the blanks one word at a time, because if we fill with one letter at a time, we might have some cases where the combination of letters do not give a meaningful word or the length of the word didn't fit in the blanks etc.

b) Formulation as a constraint satisfaction problem:
Variables: The blanks in the grid

Domains: The dictionary.
Constraints: Already placed letters must be preserved.
State: Different states of the grid - empty, partially filled in multiple ways, complete grid.
Here, we may use the Most constrained values or minimum remaining values heuristic for filling up the grid.

Using the words instead of letters would make more sense, because it will be easier to formulate the constraints with words rather than letters.

c) CSP might be a better formulation because in this case, the solution path is not so important. Rather, there is a clear set of constraints which must be satisfied. The number of constraints increase as we proceed in solving the problem.

Problem 4:
Refer the scanned images *Problem4-part1.jpg* and *Problem4-part2.jpg*

Problem 5:
a) Refer the the scanned image *Problem5a.jpg* for the complete game tree.

b) Refer the the scanned image *Problem5b-part1.jpg* and *Problem5b-part2.jpg* for the backed-up MinMas values.
There are only two possible values the nodes marked with "?". So, we can assign each of the two values to that node in turns and derive the corresponsing MinMax values.

c)The standard MinMax algorithm would fail becaseu wedon't know the MinMax values at the nodes marked with "?". As we discussed earlier, we can check in turns with all possible MinMax values for the node and obtain the MinMax values of the other nodes in the game tree.
As long as we know all the possible values which the node marked with "?" can take up, our algorithm should work with games with loops.

d) We find that the minimum number of squares possible to play such a game is 3. When there are 3 squares, B wins. But when there are 4 squares, A wins. If we try with multiple such puzzles with different number of squares greater than 2, we will find that any such puzzle can be reduced to either a 3 square game or a 4 square game. All such even number square games can be reduced to 4 square game and

allsuch odd number square games can be reduced to 3 square game.
Hence when n is even, A will win and when n is odd B will win.

Problem 6:
a) b is a Min node. Hence MinMax value at node b = minimum of Minmax values of all children = $min\{4, 14, 6\} = 4$.
h is a Max node. Hence MinMax value at node h = maximum of Minmax values of all children = $max\{1, 3\} = 3$.
i is a Max node. Hence MinMax value at node i = maximum of Minmax values of all children = $max\{6, 15, 8\} = 15$.
j is a Max node. Hence MinMax value at node j = maximum of Minmax values of all children = $max\{3, 4\} = 4$.
k is a Max node. Hence MinMax value at node k = maximum of Minmax values of all children = $max\{3, 9\} = 9$.
l is a Max node. Hence MinMax value at node l = maximum of Minmax values of all children = $max\{4, 12\} = 12$.
c is a Min node. Hence MinMax value at node c = minimum of Minmax values of all children = $min\{h, i, j\} = min\{3, 15, 4\} = 3$.
d is a Min node. Hence MinMax value at node d = minimum of Minmax values of all children = $min\{k, l\} = min\{9, 12\} = 9$.
a is a Min node. Hence MinMax value at node a = maximum of Minmax values of all children = $max\{b, c, d\} = max\{4, 3, 9\} = 9$.

The best sequence of moves at the max node a is **{a-d-k-t}**.
The utility is 9.

b) With alpha-beta pruning, the only difference we find is that the children i and j of node c get pruned i.e. they are not visited. This reduces the total runtime of the algoithm.
The nodes are visited in this order **{a-b-e-f-g-c-h-m-n-d-k-t-u-l-v-w}**.

Problem 7:
a) Dominant Strategies:
P1:
If player $P2$ play strategy a, then 2 is a better choice for $P1$.
If player $P2$ play strategy b, then 2 is a better choice for $P1$.
Hence 2 is a dominant strategy for $P1$.

P2:
If player $P1$ play strategy 1, then both a and b are same for $P2$.
If player $P1$ play strategy 2, then b is a better choice for $P2$.
If player $P1$ play strategy 3, then b is a better choice for $P2$.
Hence b is a dominant strategy for $P2$.

b) Nash Equilibrium:
The Nash equilibrium of the pay-off matrix is $(2, 3)$. If $P1$ changes his strategy from 2 to either 1 or 3, his utility decreases(when $P2$ doesn't change his strategy) from 2 to 1 in both cases. If $P2$ changes his strategy from b to a, his utility decreases(when $P1$ doesn't change his strategy) from 3 to -1. Hence, here no player gains anything by changing his/her position alone.

Problem 8:
a) Conversion of the problem into knowledge base:
System is armed: **R**
System is in motion: **M**
Alarm sounds: **A**
There is a fire: **F**

If the system is armed and motion is detected, then the alarm will sound.
If the alarm sounds, then the system has been armed or there has been a fire.
Regardless of whether the system is armed, the alarm should go off when there is a fire.
Motion is constantly detected.

Knowledge Base:
R_1: (R $\wedge$ M) => A
R_2: A => (R $\vee$ F)
R_3: F => A
R_4: M

b) *The alarm will sound if and only if the system is armed or there is a fire*
i.e. A <=> (R $\vee$ F)

Proof:
R_5: R => A using R_1 and R_4
R_6: (R $\vee$ F) => A using R_3 and R_5

R_7: A <=> (R $\vee$ F) using R_2 and R_6 [Bidirectional elimination]