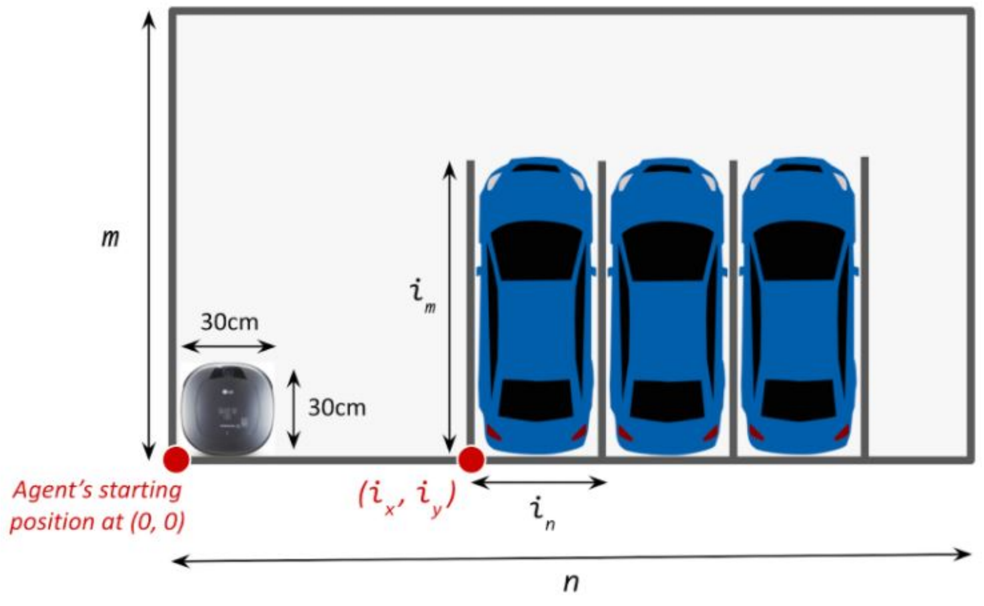
National University of Singapore
School of Computing
CS3243 Introduction to AI

Midterm Examination Review (Solutions)

1. You are tasked to design a floor-cleaning agent for a carpark.

   Assume that the problem environment is a rectangular carpark that is enclosed, i.e., all entrances/exits are sealed and the agent cannot leave the carpark. The size of the carpark is $m \times n$, where length m refers to the distance, in centimetres (cm), in the North direction starting from the agent's starting position, and width n refers to the distance, in centimetres (cm), in the East direction.

   Assume further that the floor-cleaning agent is of size $30cm$ by $30cm$ and that it starts at the bottom-left-most corner of the carpark. The figure below shows an example of one such carpark with the floor-cleaning agent in its starting position (image may not be drawn to scale).



   The agent can only move in four directions: Up, Down, Left and Right, with a specified (non-zero) distance. At each position, the agent can also choose to clean the area directly beneath it, i.e., the $30cm$ by $30cm$ area that it currently occupies.

Assume further that the carpark contains k rectangular parking spaces that are orthogonal to the perimeter of the carpark (as shown in the figure above, the lines defining each rectangular parking space are parallel to the lines defining the perimeter of the rectangular carpark).

For each parking space i (where $1 \leq i \leq k$), the size ($i_m \times i_n$) and position (i_x, i_y) of the space are known. Length i_m refers to the distance, in centimetres (cm), in the North direction starting from the bottom-left-most point of the parking space, and width i_n refers to the distance, in centimetres (cm), in the East direction. Further, i_y refers to the displacement, in centimeters (cm), of the bottom-left-most point of the parking space in the North direction relative to the agent's starting position, and i_x refers to the displacement, in centimetres (cm), in the East direction. The starting position of the agent is $(0, 0)$.

Assume that all parking spaces in the carpark are occupied by vehicles and that the agent cannot access these spaces (occupied areas). You may assume that the environment is static, i.e., the vehicles will not move. Also assume that either the front, back, or both front and back, of all parked vehicles are not obstructed by other vehicles.

The goal of the agent is to clean all the accessible areas in the carpark, i.e., all areas not occupied by the parking spaces (unoccupied areas). You may assume that all areas not occupied by the parking spaces are accessible by the agent.

Formulate the above as a general search problem. More specifically, define the following:

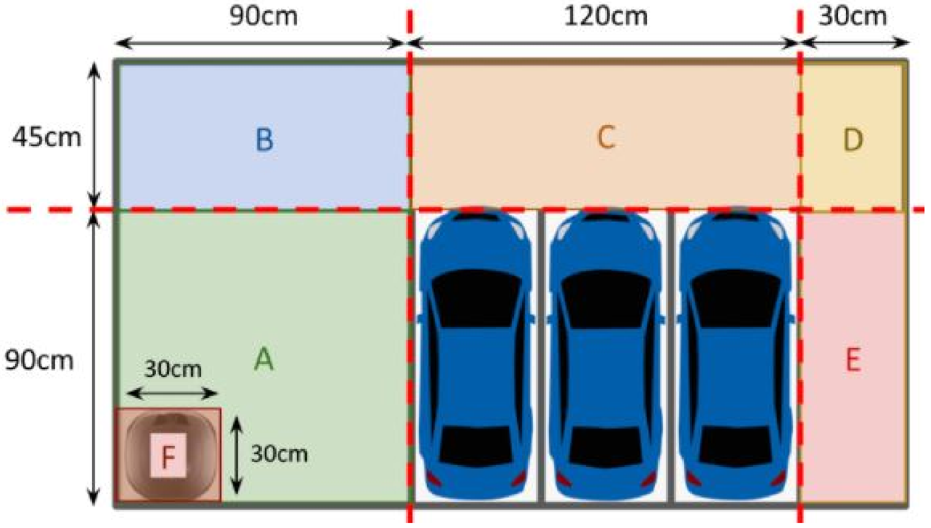
- State representation
- Initial state
- Actions
- Transition model
- Step cost
- Goal test

If necessary, you may identify any other assumptions you have made. However, assumptions that are contradictory to any instruction in the question, or that are unreasonable, will be invalid.

Solution: This problem has a continuous space and hence, the key idea of the formulation is to discretize the problem so that the search space is not infinite.

One ideal formulation is to first determine the Greatest Common Divisor (GCD) of unit area ($GCD_{area} = GCD_m \times GCD_n$) across all rectangular unoccupied areas in the carpark that has to be cleaned by the agent as well as the (rectangular) area that the agent occupies (we need to consider this as the agent can only clean the area beneath it at each time step).

In the simple example below, the rectangular unoccupied areas are given by blocks A, B, C, D, and E, while the area that the agent occupies is given by block F. To determine GCD_{area}, we first determine GCD_m and GCD_n based on the length b_m and width b_n parameters of these blocks $b \in \{A, B, C, D, E, F\}$. Recall that length m refers to the distance in the North direction and width n refers to the distance in the East direction. Using the distances labelled in the figure below, we get $GCD_m = 15cm$ and $GCD_n = 30cm$, and hence $GCD_{area} = 450cm^2$. We then use these values in our state representation.



Note that this formulation is modelled after a path planning problem, where we are trying to determine a path that the agent would take to reach its goal (to completely clean a given carpark).

- **State representation**
  - 2D array s of size m/GCD_m by n/GCD_n cells. Each cell represents a carpark area of $GCD_m \times GCD_n = GCD_{area}$. GCD_m, GCD_n and GCD_{area} are computed as explained above.
  - Cell values in s:
    - ”0”: Unoccupied area and not cleaned
    - ”1”: Unoccupied area and cleaned
    - ”X”: Occupied area (parking spaces)
  - Current agent position $p = (p_x, p_y)$ whose x and y coordinates refer to the cell indices in s.

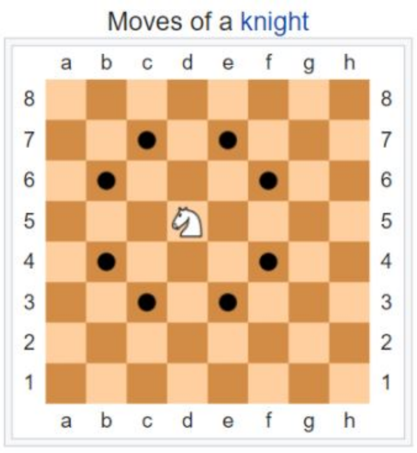
- **Initial state**
  - All cells in s are either “0” or “X” (all cells that are “0” need to be cleaned). The cells representing the areas occupied by parking spaces are marked as “X” while the other cells representing unoccupied areas are marked as “0”.
  - Starting agent position $p = (0, 0)$.
- **Actions**
  - Up by a distance of GCD_m
  - Down by a distance of GCD_m
  - Left by a distance of GCD_n
  - Right by a distance of GCD_n
  - Clean (the area beneath the agent)
  - Legal movements are also determined based on the agent’s current position in the grid and the obstacles/walls around it. E.g., at the agent’s starting position (bottom-left-most corner of the carpark), the agent cannot move Down and Left (walls). And if there are occupied cell(s) beside it, the agent also cannot move in the direction of the occupied cell(s) (obstacles).
- **Transition Model**
  - Given matrix s, position p and action a,
    - If a = Up/Down/Left/Right, update p to reflect the new agent position. E.g., if a = Up, $p' = (p_x, p_y + 1)$.
    - If a = Clean, update s to assign the values of the cells representing the area currently occupied by the agent to “1”.
    - Specifically, a total of $(agent_{area}/GCD_{area})$ cells will be assigned a value of “1” in s':

      (p_x, p_y), $(p_x + 1, p_y)$, $(p_x, p_y + 1)$, $\ldots$,
      $(p_x + (agent_n/GCD_n), p_y + (agent_m/GCD_m))$,

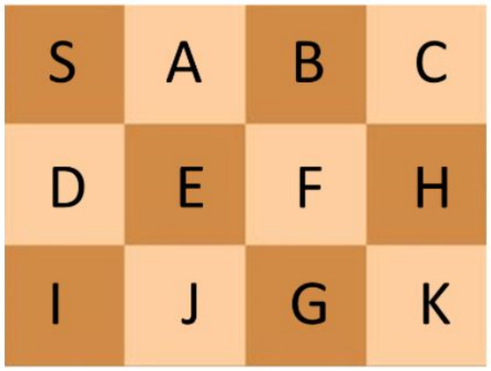
      where $agent_m$ and $agent_n$ refer to the length and width of the agent respectively.
- **Step cost**
  - 1 (or any positive constant)
- **Goal test**
  - Number of “1” cells = (total carpark area − total parking space area) $/GCD_{area}$

2. Consider a chess-variant game, where we are given a Knight on a 3 board, with its initial position labelled as S and its goal position labelled as G.

   Refer to the picture below on the moves that a Knight can make (which you should be familiar with by now given Project 1 and 2).

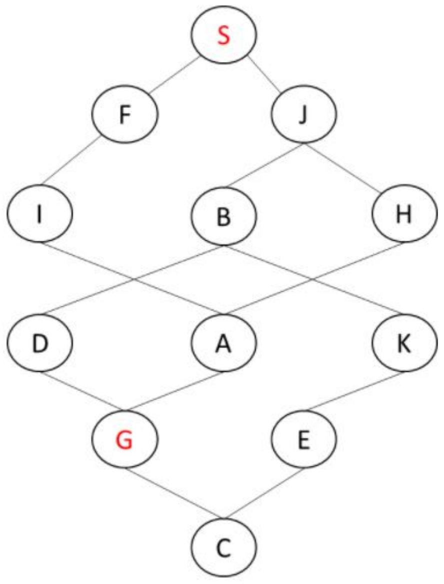


   In the table below, you are given the (3) board configuration, with letters labelling each grid in the board. As mentioned above, the Knight is placed at the starting position S and the goal position is labelled as G.



S	A	B	C
D	E	F	H
I	J	G	K

The undirected search graph for the Knight is as follows.



(a) Trace the following algorithms. With each trace, specify the order of the nodes in terms of when the goal test is performed (i.e., first node tested, then second, then third, etc.). Assume that when pushing to the frontier, ties (i.e., when pushing multiple successors) are broken based on alphabetical order.

i. BFS with limited-graph-based implementation and early goal test.

ii. BFS with limited-graph-based implementation and late goal test.

iii. DFS with limited-graph-based implementation (note that DFS does not use an early goal test).

(b) Assume a heuristic h, such that the heuristic value, $h(x)$, for each grid position, x, is shown below. Additionally, all step costs have a value of 1.

$S, h(S) = 3$	$A, h(A) = 1$	$B, h(B) = 1$	$C, h(C) = 1$
$D, h(D) = 1$	$E, h(E) = 2$	$F, h(F) = 2$	$H, h(H) = 2$
$I, h(I) = 1$	$J, h(J) = 1$	$G, h(G) = 0$	$K, h(K) = 3$

i. Trace the A* Search with limited-graph-based implementation using h by specifying the order of the nodes in terms of when the goal test is performed (i.e., first node tested, then second, then third, etc.). Assume that when pushing, or popping, from the frontier, ties (i.e., nodes with the same evaluation function value) are broken based on alphabetical order.

ii. Using A* search with limited-graph-based implementation using h, determine the path taken by the Knight from its starting position S to the goal position G.

Note that you should express your answers in the form S–B–A–F–G (i.e., no spaces, all uppercase letters, delimited by the dash (–) character), which, for example, corresponds to the order S, B, A, F, and G.

Solution:

(a) i. S-F-J-I-B-H-A-D-K-A-G

ii. S-F-J-I-B-H-A-D-K-A-G

iii. S-J-H–A-I-F-G

(b) i. S-J-B-F-I-A-D-G

ii. S-F-I-A-G or S-J-B-D-G

3. Let C^* be the cost of an optimal path. Assuming that every action costs at least some small positive constant ϵ. Explain why the time and space complexity of the UCS algorithm is given as: $O(b^{1+\lfloor C^*/\epsilon \rfloor})$.

Solution:

- In the worst case, we assume step costs of all actions are ϵ. Since, at each state, the number of possible actions is b, we may assume the progression:

$$1 \text{ (initial state)} + b \text{ (nodes with path cost } \epsilon) +$$
$$b^2 \text{ (nodes with path cost } 2\epsilon) + \ldots + b^{\lfloor C^*/\epsilon \rfloor}$$

- However, note UCS performs a late goal test, which means that by the time it checks the node at depth $\lfloor C^*/\epsilon \rfloor$, it would have also generated, in the worst case, all the nodes in the following level as well, which requires an additional term $b^{\lfloor C^*/\epsilon \rfloor + 1}$.
- This series sums to $(b^{\lfloor C^*/\epsilon \rfloor + 2} - 1)/(b-1) = O(b^{\lfloor C^*/\epsilon \rfloor + 1})$.
- It should also be noted that we take the floor since step size is lower bounded by ϵ - i.e., since we cannot have a step cost $< \epsilon$, the remainder from C^*/ϵ must come from some steps costs in the optimal path having values higher than ϵ, and not from one additional step that had cost $< \epsilon$.

4. Suppose you have two admissible heuristics, h_1 and h_2. You are also given an inadmissible heuristic h_3. You decide to create the following new heuristic functions:

   - $h_4(n) = min(max(2 * h_1(n), h_3(n)), h_2(n))$
   - $h_5(n) = max(h_1(n), min(h_2(n), h_3(n)))$
   - $h_6(n) = min(h_3(n), max(h_1(n), 1.1 * h_2(n)))$
   - $h_7(n) = max((h_1(n) + h_2(n))/2, h_3(n))$

   For each of the new heuristics, specify if it is admissible or not. Justify your answer.

   Solution:

   - h_4 is admissible. The minimum of admissible and inadmissible is admissible.
   - h_5 is admissible. The maximum of admissible and admissible is admissible
   - h_6 is indeterminant. When $h_2(n) = 0$ for all n, h_6 is admissible. However, when $h_2(n) = h^*(n)$ for all n, h_6 is inadmissible.
   - h_7 is inadmissible. The maximum of inadmissible and admissible is inadmissible.

5. Given a set of admissible heuristics: h_1, h_2, h_3, $\ldots$, h_n, define a new heuristic g where g is a function of the set of the heuristics that will result in A* expanding a minimal number of nodes while still guaranteeing admissibility.

   Solution: $g(n) = max(h_1(n), h_2(n), h_3(n), \ldots, h_n(n))$, for all states n.

   Using just 2 heuristics as an example:

   Consider the true cost $h^*(s)$. For admissibility, both $h_1(s) \leq h^*(s)$ and $h_2(s) \leq h^*(s)$, and their max can be no larger than either. $h^*(s)$ leads to expanding the minimal number of nodes; in particular it expands the nodes of an optimal path to a goal and no more. g is the closest heuristic to h^* of guaranteed admissibility by h_1 and h_2 and so expands a minimal number of nodes for functions over h_1, h_2.

6. UWordle is a word game where players attempt (with unlimited guesses) to guess a 5-letter English target word. After every guess, feedback will be in the sequence of 5 colored tiles corresponding to the characters in the guessed word, indicating:

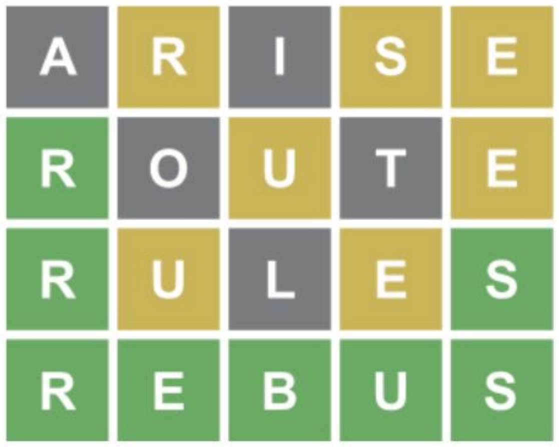
   - Green - Letter is in the word and in the correct position.
   - Yellow - Letter is in the word but in the wrong position.
   - Grey - Letter is not in the word at all.

   Note that in the original game (i.e., Wordle), there are only 6 guesses; our version, UWordle has unlimited guesses.

   Each word guessed, as well as the target word, are valid English words of 5-letters. (Also note that for the purposes of the UWordle problem, only consider capitalised English letters of the alphabet.)

   The following example illustrates a game of UWordle completed in 4 guesses. The goal is to guess the word "REBUS". The first guess of "ARISE" yielded the following information:

   - The characters A, I are not in the target word.
   - The characters R, S, E are in the target word but not in the correct positions.



   This question is concerned with solving the UWordle puzzle as a search problem.

Assume we have a valid dictionary, D, of all valid 5-letter English words, and that the target word can be found in D. Given a fixed but unknown target word, we represent the search problem as follows.

- **State Space**: Each state is represented by the set of candidate words C_i, where C_i is the set of possible target words (which will include the actual target word) for that state.
- **Initial State**: Entire dictionary D.
- **Goal State**: The size of candidate words $|C| = 1$.
- **Action**: Pick a word in D.
- **Transition Model**: From the observation (i.e., sequence of colored tiles received as feedback) following each action, we exclude all words in C_i that violate the new observed constraints to create C_{i+1}.
- **Cost**: The cost of every guess is 1.

(a) Determine the environment characteristics of the above problem.

i. Fully or Partially Observable?

ii. Single or Multi-Agent?

iii. Deterministic or Stochastic?

iv. Static or Dynamic?

v. Discrete or Continuous?

(b) Heuristic h_1 is such that for any state s_i, $h_1(s_i) = k$, where k is a constant value. State and prove that for your chosen value of k, h_1 is admissible.

(c) Propose another distinctly different but useful[1] heuristic h_2 that dominates h_1. Explicitly state h_2, explain how h_2 can be computed and proof dominance.

Discuss how h_2 is useful to a local search algorithm.

If your proposed heuristic is admissible, provide a proof. If your proposed heuristic is not admissible, proof and justify why another admissible heuristic cannot be found.

[1] Similar to Tutorial 3, Q1, your heuristic must be computable, explicitly defined and non-trivial (i.e., your heuristic may not be $h(s) = 0$ for all states s, the (abstract) optimal heuristic, $h*$, or a linear combination/simple function thereof).

Solution:

(a) i. Partially Observable. The next state cannot be observed until action is taken.

ii. Single Agent. The target word is set beforehand and cannot be changed.

iii. Deterministic. The next state is determined by action and the current state since the target word is fixed.

iv. Static. The target word does not change.

v. Discrete. Finite actions and finite states.

Note that this is a known problem, i.e., all the rules are known but the target word is hidden. Also, note that the current action can affect future decisions, but only indirectly - i.e., it may be argued to either be Sequential or Episodic.

(b) $k = 0$.

$h_1(s_i) = k$

$h^*(g) = 0$, $h^*(s_i) \geq 0$ [goal state has the smallest value]

$h_1(s_i) \leq h^*(s_i)$ [admissibility]

$k \leq h^*(s_i)$

The only valid k is 0.

(c) **State h_2, explain how h_2 can be computed and prove dominance**

For all s_i, $h_2(s_i)$ can be the expected ratio of words left at s_i when compared to the dictionary D. Note that while s_i is unknown, we do know the following.

$$h_2(s_i) = \mathbb{E}\left[\frac{|C_i|}{|D|}\right]$$

h_2 of the next state can be computed by generating all possible observations conditioned on s_{i-1}.

At s_{i-1}, we know the set of candidate words C_{i-1} after all previous observations.

For the next s_i, we can apply all $\{Green, Yellow, Grey\}^5$ possible observations from C_{i-1} to get 3^5 different but possible C_i, each denoted $C_i^{(j)}$, where $1 \leq j \leq 3^5$.

Then, we compute the average over all possible ratios:

$$h_2(s_i) = \mathbb{E}\left[\frac{|C_i|}{|D|}\right] = \mathbb{E}\left[\forall_j \frac{|C_i^{(j)}|}{|D|}\right]$$

Note that the smallest h_2 is when C_i is the smallest. C_i is the smallest when the observation is all Greens, indicating the correct word, with one observation. In that case,

$$min\,[h_2(s_i)] = 1/|D| > 0 = h_1,$$

where $h_1(s_i) = 0$ for all states s_i. So, h_2 dominates h_1.

Explain how h_2 is useful

It is useful as it guides the search greedily (1 step lookahead) towards the choice of guess where it maximises the removal of words from the current candidate space via choosing the state with the lowest expected ratio.

Proof that there cannot be any other admissible heuristics

h_2 is inadmissible.

> Let s_i be the goal state, then $h^*(s_i) = 0$, but $h_2(s_i)$ is non zero.
> Assume an admissible heuristic h'.
> At the current state, the target word can be the action.
> If the agent takes the action, the goal state will be reached.
> The $h^*(g) = 0$, and consequently $h'(g) = 0$ for admissible heuristic.
> For $h'(g) = 0$, h' must have known g is the goal state or $h' = 0$.
> But h' does not have access to the exact next state but only an estimate.
> So, h' cannot be admissible unless $h' = 0$.

There are no other admissible heuristics, including h_2.