NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

MIDTERM ASSESSMENT FOR
Semester 2 AY2023/2024

CS3243: INTRODUCTION TO ARTIFICIAL INTELLIGENCE

SOLUTIONS

March 4, 2024 Time Allowed: 90 Minutes

INSTRUCTIONS TO CANDIDATES

1. This assessment contains THREE (3) questions. All the questions are worth a total of 30 MARKS. It is set for a total duration of 90 MINUTES. You are to complete all 3 questions.
2. This is a CLOSED BOOK assessment. However, you may reference a SINGLE DOUBLE-SIDED A4 CHEAT SHEET.
3. You are allowed to use NUS APPROVED CALCULATORS.
4. If something is unclear, solve the question under reasonable assumptions. State your assumptions clearly in the answer. If you must seek clarification, the invigilators will only answer questions with Yes/No/No Comment answers.
5. You may not communicate with anyone other than the invigilators in any way.

STUDENT NUMBER:	**A**								

EXAMINER'S USE ONLY		
Question	Mark	Score
1	10	
2	10	
3	10	
TOTAL	30	

1. Suppose that you are given an acyclic graph, $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$, comprised of (1) a set of distinct vertices, $\boldsymbol{V} = \{v_1, \ldots, v_n\}$, where each $v_i \in \boldsymbol{V}$ corresponds to a unique state, and (2) a set of distinct directed edges, $\boldsymbol{E} = \{e_1, \ldots, e_m\}$, where each $e_j \in \boldsymbol{E}$ corresponds to a 3-tuple, (a, b, c_j), which connects vertex v_a to vertex v_b with cost c_j. Further, assume that $v_s \in \boldsymbol{V}$ is the initial state for $\boldsymbol{G}$, and that you are given a goal test function ***check***(v), which takes a state v as input and returns *True* if v is a goal state, otherwise, returns *False*.

We wish to solve a search problem where the goal is to define the optimal heuristic for $\boldsymbol{G}$, h*, such that for $v_i \in \boldsymbol{V}$, we have:

$$h^*(v_i) = \begin{cases} 0 & \text{if } \textbf{\textit{check}}(v_i) \\ t & \text{if } \neg\textbf{\textit{check}}(v_i) \text{, where } t \text{ is the lowest path cost from } v_i \text{ to a goal state} \end{cases}$$

Assume that you have the following partial search problem formulation for the problem described above.

- ***Pre-processing***:
  - Generate a global ***hash table***, ***H***, which is keyed on vertices $v_i \in \boldsymbol{V}$. Each $\boldsymbol{H}[v_i]$ corresponds to the value $h^*(v_i)$. Initially, all values are assigned the value x.
- ***State representation***:
  - Each state is represented by a ***frontier***, $\boldsymbol{F} = \{\boldsymbol{N}_1, \ldots\}$, which corresponds to a ***stack*** data structure. Each $\boldsymbol{N}_i \in \boldsymbol{F}$ corresponds to a ***node*** abstract data type that will be used to facilitate a search of $\boldsymbol{G}$.
- ***Cost function***:
  - All actions have uniform cost $k \geq 0$.

(i) [0.5 marks] What should we use as the value x?

Solution: $x = \infty$ (Any token value indicating a previously unvisited node is also acceptable.)

(ii) [0.5 marks] Given that a ***stack*** is adopted as the ***frontier F***, which search algorithm is being implemented to solve ***G***?

Solution: Depth-first search (DFS) or iterative-deepening search (IDS).

(iii) [1.0 marks] What should be the composition of each ***node*** $\boldsymbol{N}_i$ (i.e., the attributes of the *node* abstract data type) that will be stored on the ***frontier F***?

Solution: Each $\boldsymbol{N}_i$ corresponds to the 3-tuple $(v_i, \boldsymbol{N}_p, c_q)$, which contains:

- v_i, the last state on the path representing $\boldsymbol{N}_i$;
- $\boldsymbol{N}_p$, the parent node (thus, the next link in the linked list representing path $\boldsymbol{N}_i$); and
- c_q, the path cost of the path represented by $\boldsymbol{N}_i$.

(iv) [1.0 marks] Determine if a ***tree search*** or ***graph search*** implementation should be adopted to solve ***G***.

Solution: **Tree Search**. /* The problem requires the traversal of all paths in ***G***, which precludes the use of Graph search. Thankfully, since ***G*** is acyclic and finite, tree search is complete. */

(v) [6.5 marks] Complete the given search problem formulation by specifying the following.

- ***Initial state***.
- ***Actions***.
- ***Transition model***.
- ***Goal test***.
- (Optional) additional ***pre-processing***.

Note that your search problem formulation should be efficient.

Solution: The search problem formulation is as follows.

- ***Pre-processing***: In addition to the global ***hash table***, $\boldsymbol{H}$, with initial values specified in **(i)**, we also have the following.
  - Generate a global ***hash table*** $\boldsymbol{V'}$, that is keyed using states, v_i, such that $\boldsymbol{V'}[v_i]$ returns the set of edges $\boldsymbol{E'} = \{e_a, \ldots\}$, where for each $e_j \in \boldsymbol{E'}$, $e_j[0] = v_i$ (i.e., each $e_j \in \boldsymbol{E'}$ has the source vertex v_i), $e_j[1]$ is a destination vertex, and $e_j[2]$ is the cost corresponding to the traversal from $e_j[0]$ to $e_j[1]$.
- ***State representation***: As given, with node composition specified in **(iii)**.
- ***Initial state***:
  - Push the node $(v_s, None, 0)$ onto $\boldsymbol{F}$.
- ***Actions***:
  - For each state, there is only one possible action, which is to perform one additional iteration of the search of $\boldsymbol{G}$ via $\boldsymbol{F}$.
- ***Transition model***:
  - Given the frontier $\boldsymbol{F}$ for a parent state, and the action to perform one additional iteration of the search of $\boldsymbol{G}$, then we must perform the following.
    - Pop $\boldsymbol{N}_i$ from $\boldsymbol{F}$ (i.e., pop the top of the frontier (i.e., the stack)).
    - If $\neg$***check***$(\boldsymbol{N}_i[0])$:
      - Generate $\boldsymbol{E'} = \boldsymbol{V'}[\boldsymbol{N}_i[0]]$.
      - For each e_p in $\boldsymbol{E'}$, generate the node $\boldsymbol{N}_p$, where:
        - $\boldsymbol{N}_p[0] = e_p[1]$ // new path; successor as new last state.
        - $\boldsymbol{N}_p[1] = \boldsymbol{N}_i$ // link to parent node.
        - $\boldsymbol{N}_p[2] = \boldsymbol{N}_i[2] + e_p[2]$ // new path cost.
        - And push $\boldsymbol{N}_p$ onto $\boldsymbol{F}$.
    - Else (if ***check***$(\boldsymbol{N}_i[0])$ – i.e., $\boldsymbol{N}_i[0] = v_i$ is a goal state):
      - Update $\boldsymbol{H}[\boldsymbol{N}_i[0])]$ to 0.
      - Let $current\_node = \boldsymbol{N}_i[1]$
      - While $current\_node$!= $None$:
        - If $\boldsymbol{N}_i[2] - current\_node[2] < \boldsymbol{H}[current\_node[0]]$:
          - $\boldsymbol{H}[current\_node[0]] = (\boldsymbol{N}_i[2] - current\_node[2])$

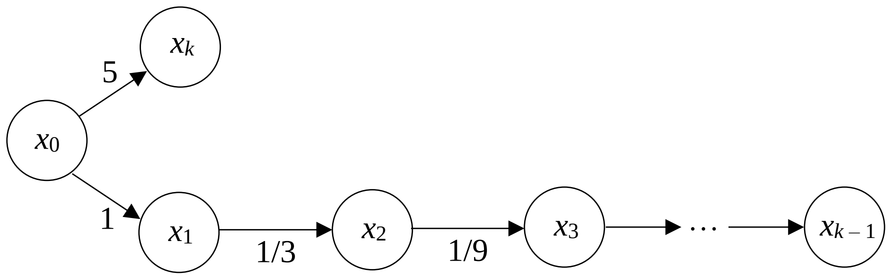
            /* Note that $\boldsymbol{N}_i[2] - current\_node[2]$ is the path cost from $v_j = current\_node[0]$ to $v_i = \boldsymbol{N}_i[0]$ (recall that v_i is a goal). */
        - Let $current\_node = current\_node[1]$ (i.e., update $current\_node$ to the next node in the linked list, which is the parent of $current\_node$).
- ***Goal test***:
  - Simply return False – i.e., there is no goal test since we must complete an entire traversal of $\boldsymbol{G}$ to ensure we have checked all paths – i.e., $\boldsymbol{F}$ is empty.

// Note: defining a reverse search from goal states breaks "…facilitate a search of $\boldsymbol{G}$".

(vi) [0.5 marks] Determine the space complexity (i.e., number of *nodes* generated) for the resultant search tree (i.e., to determine h*).

Solution: O(|***V***| + |***E***|)

2a. Consider the following state space. The initial state is x_0 and the goal state is x_k. The action to state x_i always has cost $1/3^{i-1}$, except the action that leads to the goal state.



(i) [1.0 marks] Trace the nodes that are popped by the **Depth-First Search** (**DFS**) algorithm, assuming that it is implemented using **tree search**. You must further assume that nodes are pushed onto the frontier based on ascending numerical ordering of the subscripted index (e.g., x_2 would be pushed before x_3), and that k approaches ∞.

Solution: (x_0) – (x_k)

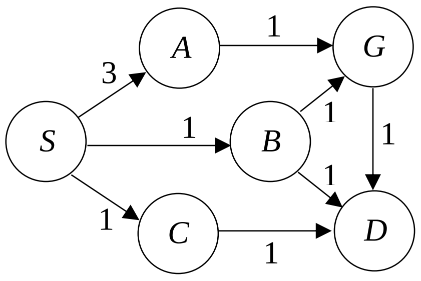
(ii) [1.0 marks] Trace the nodes that are popped by the **Breadth-First Search** (**BFS**) algorithm, assuming that it is implemented using **tree search**. You must further assume that nodes are pushed onto the frontier based on ascending numerical ordering of the subscripted index (e.g., x_2 would be pushed before x_3), and that k approaches ∞.

Solution: (x_0) // due to early goal test

(iii) [1.0 marks] Would the **Uniform-Cost Search** (**UCS**) algorithm be complete when applied to the given graph? Assume that k approaches ∞. Explain your answer.

Solution: UCS is **not complete** for the given graph. The reason is that the action costs in the graph are not bounded by a positive number (i.e., ε). For this particular graph, the path cost sums to a number that is less than 5. Hence, ***G*** will never be explored.

2b. Consider the following state space. The initial state is *S* and the goal state is *G*. The cost of each action is reflected on each edge.



n	h(*n*)
S	0
A	5
B	4
C	3
D	7
G	0

(i) [1.0 marks] Is the specified heuristic, h, admissible?

Solution: No. For states *A*, *B*, *C*, and *D*, h(*n*) > h*(*n*).

(ii) [2.0 marks] Specify the path returned by the A* algorithm, assuming that it is implemented using Graph Search Version 3.

Solution: *S* – *B* – *G*

(iii) [1.0 marks] Is the path given in Part (ii) optimal?

Solution: Yes.

2c. [3.0 marks] Prove or disprove the following statement.

*The **Iterative Depth-First Search** (**IDS**) algorithm, implemented using Graph Search Version 1, is optimal under finite graphs.*

Solution: False.

Suppose that we have a graph that has exactly two paths to the goal: (1) S – A – B – D – G, and (2) S – C – D – G. Assume tie-breaking is done based on reverse alphabetical ordering (i.e., B before A).

If we implement IDS using DLS with Graph Search Version 1, the trace would be as follows.

- Depth limit 0: expand S
- Depth limit 1: expand S, expand A, expand C
- Depth limit 2: expand S, A, B, C, D
- Depth limit 3: expand S, A, B, D, C // terminate, since D has been visited
- Depth limit 4: expand S, A, B, D, G // terminate, since goal found

The issue is that the path S – C – D can never be explored since by the time C is expanded, D is already in the reached set (from the expansion of B leading to D being inserted into the reached set). So, the optimal path S – C – D – G cannot be found.

3. In computational biology, there are often times when the similarity of two strings of DNA must be compared. Furthermore, the series of transformations to convert one string into another can often contain information regarding the various mutations that have occurred. In this question, we will explore different algorithms that model and solve such string similarity comparisons.

More specifically, this problem is defined as follows. Given a source string, the objective is to apply a series of actions such that this source string is transformed into a given target string. Note that the target string need not be the same length as the source string. For any given string, the following three types of actions are possible.

i. Inserting a character at any position in the string.
ii. Deleting any character in the string.
iii. Substituting one character in the string for any other allowed character.

You may assume all three actions have uniform cost of 1.

For example, to convert the string "ATAT" to "CATO", we may apply the following 3 actions.

- Insert a "C" at index 0 of "ATAT", hence yielding "CATAT".
- Remove the "A" at index 3 of "CATAT", hence yielding "CATT".
- Substitute the "T" at index 3 of "CATT" with an "O", hence yielding "CATO".

A student wishes to solve this problem using local search. A heuristic is defined to compare a source string to a target string. To do so, we count the number of times the characters at the same index of both strings mismatch. If one string is longer than the other, then consider the extra characters in the longer string as one mismatch each.

For example, given a target string "SCHOOL" and the evaluated string "SPOOL", the heuristic specified above would output 4. The following is the justification.

Index	0	1	2	3	4	5
Evaluated string	S	P	O	O	L	[BLANK]
Target string	S	C	H	O	O	L
Is Different?	0	1	1	0	1	1
Total difference = 0 + 1 + 1 + 0 + 1 + 1 = 4						

(i) [1.0 marks] Given a target string "GOAL", determine the output of the specified heuristic when given the source string "SEARCH".

Solution: ***eval***("SEARCH") = 5

(ii) [3.0 marks] Prove the completeness of the hill-climbing algorithm if the given heuristic is used as the evaluation function. You may assume the variant of hill-climbing with no sideways moves and no random restarts.

Solution:

If $\boldsymbol{eval}(s) = 0$, then s is the target string t. Or else if $\boldsymbol{eval}(s) > 0$, then one of the following is always possible:

- If $\boldsymbol{len}(s) < \boldsymbol{len}(t)$, insert at the end of s the character in the corresponding position of t.
- If $\boldsymbol{len}(s) > \boldsymbol{len}(t)$, delete the last character in s.
- If $\boldsymbol{len}(s) = \boldsymbol{len}(t)$, there is at least one index where the character in s and t differ. Substitute that character in s for the corresponding character in t.

In all 3 cases above, heuristic value of the updated string s' decreases by 1.

As the evaluation of any string is always finite, and it is always possible to find an action that decreases the heuristic value, the hill-climbing algorithm must, at some point, reach a state where the heuristic value is 0, at which point the search is complete.

To obtain a path that transforms a source string into the target string, another student suggests that one may use a path search algorithm such as the A* search algorithm. The student further suggests that the previously defined heuristic would also be a good choice for the A* search algorithm.

(iii) [2.0 marks] Prove or disprove the admissibility of the previously defined heuristic, when it is utilised with the A* search algorithm.

Solution:

The previous heuristic is inadmissible by counterexample. The string "SPOOL" requires 2 actions in order to be transformed into the string "SCHOOL" (i.e., insert a "C" after "S", and replace "P" with "H"), and thus has a cost of 2. However, the heuristic predicts a cost of 4, which represents an overestimate and is hence inadmissible.

/* Note that the heuristic is inadmissible as it represents a tightening of the problem's constraints (as opposed to desired relaxations when formulating heuristics). Instead of allowing insertions, deletions and substitutions, the heuristic only allows substitutions. */

(iv) [3.0 marks] Define an admissible heuristic for the given problem. Your heuristic must be defined mathematically. The defined heuristic must also have no upper bound, i.e., given an integer k and a target string t, there must exist some string s such that $h(s) > k$. Prove that your heuristic is admissible, and has no upper bound.

In addition to normal mathematical operations, you are also allowed to use the functions ***count***(s, c), which returns the number of occurrences of character c in string s, and ***len***(s), which returns the length of the string s.

Solution:

One possible heuristic is | ***len***(s) – ***len***(t) |, where t is the target string and s is the current string being evaluated.

When the strings differ in length by n, there are at least n deletion or insertion operations required to allow the strings to have equal length, which in turn equates to a cost of at least n. More generally, this heuristic is admissible as it represents a relaxation of the problem's constraints, by allowing substitutions to be made at 0 cost, while insertions and deletions still maintain their cost of 1.

This heuristic also has no upper bound, as given an integer k, and a target string t, any string s with ***len***$(s) \geq$ ***len***$(t) + k$ will have $h(s) > k$.

(v) [1.0 marks] Would you prefer to use local search or A* search to solve this problem? Justify your answer.

Solution: A* since the path (i.e., the sequence of mutations) is desired to define the similarity. However, local search may also be viable if it is modified to also return the path. This modification is plausible in this case since the local search is complete. Further, under local search, you would also have better space complexity.

(Note that optimality is not explicitly required.)

END OF PAPER

BLANK PAGE

BLANK PAGE