COMP3702 2021 - Practice Exam

MODULE 1: SEARCH

Question 1.1

Regarding blind search algorithms:

a) What data structure is best used to implement breadth-first-search?

A (first-in-first-out) queue

b) What blind search algorithm or algorithms can be implemented using priority queue?

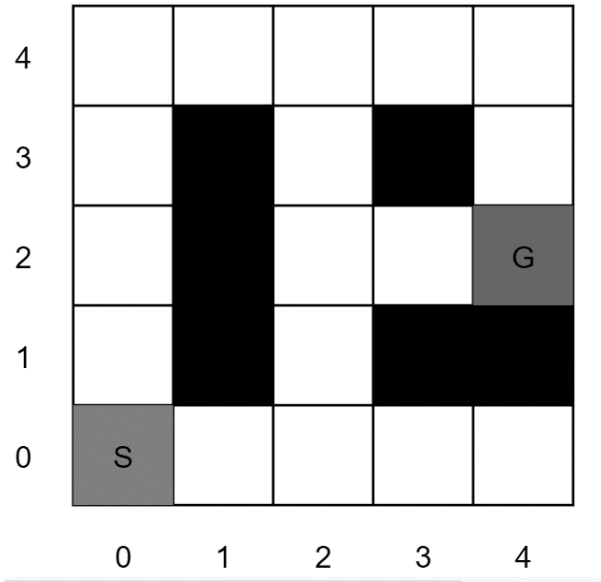
Uniform cost search (not A* search because it is not blind)

c) Explain in your own words the benefits of using iterative-deepening depth-first search over depth-first search in a search problem.

IDDFS combines the optimality and completeness of BFS with the space (memory) efficiency of DFS. Whenever the goal state is shallower than the depth of the search tree, IDDFS takes a little more time than BFS, but saves a lot of memory compared to BFS (like DFS).

Question 1.2

Answer the following questions about the search problem on the gridworld shown below:



The details of this problem are:

- The 5x5 grid has obstacles indicated by black tiles in three contiguous blocks, at coordinates (1,1), (1,2), (1,3); at (3,1) and (4,1); and at (3,3).
- The initial state is S, at coordinates (0,0), and the goal state is G, at (4,2).

For the questions that ask for a path, please give your answers in the form 'S–(x,y)–…–(x,y)–G'. Break any priority ties using the move order *up, right, down, left*.

a) What path does iterative-deepening depth-first search with depth parameter k = 3 return for this search problem?

Without a limit on the depth, IDDFS is optimal and would return:
S-(1,0)-(2,0)-(2,1)-(2,2)-(3,2)-G

Since it has a depth limit of 3, it would not return a path.

b) Now assume a constant cost of 1 per move. What path does uniform cost search return for this search problem?

S-(1,0)-(2,0)-(2,1)-(2,2)-(3,2)-G

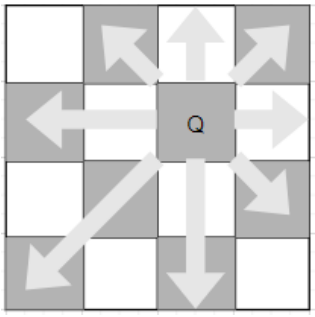
c) Consider a heuristic for this search problem given by $h = 4\text{-}x$ (x is the x-coordinate).
   i. Is the heuristic *h* consistent? Yes
   ii. Is the heuristic *h* admissible? Yes

MODULE 2: REASONING WITH CERTAINTY

Question 2.1

In chess, a queen can move in straight lines any number of squares left or right, forward or backward, or diagonally. The 8-queens puzzle is a classic problem of placing eight chess queens on an 8x8 chessboard so that no two queens threaten each other.

A simpler version of the problem is the 4 queens puzzle, played on a 4x4 board, shown below:



The possible moves of a single queen are shown in this figure.

The typical way to model this problem is to assign each of the 4 queens its own column, A, B, C or D, and then choose a row 1, 2, 3 or 4 in such a way that they cannot attack each other. Starting from this variable definition, answer the following questions on the 4 queens puzzle.

a) What is the domain of each queen?

Row numbers, i.e. $Q_i \in \{1,2,3,4\}$

b) List all binary constraints between variables for this CSP. How many constraints are there?

Row-different(Q_i,Q_j) for rows *i,j* in *{A,B,C,D}*, with *i* not equal to *j*

- I.e. $Q_i \neq Q_j$
- How many? From the left, there are three Row-diff for Q-A, two more for Q-B, and on more for Q-C; Q-D's row-diff constraint are included in the rest.
- Six row-diff constraints.

Diag-different(Q_i,Q_j) for *i,j* in *{A,B,C,D}*, *i* not equal to *j*

- They cannot be the same number of columns apart as they are rows apart, i.e. $|Q_i - Q_j| \neq |i-j|$
- How many? Same as Row-diff; for Q-A from the left, there are 3 Diag-diff for Q-A, two more for Q-B, and on more for Q-C; Q-D's row-diff constraint are included in the rest.
- Six row-diff constraints.

12 binary constraints in total.

c) Express the problem as one of logical satisfiability in *conjunctive normal form*.

$(Q_A \neq Q_B) \wedge (Q_A \neq Q_C) \wedge (Q_A \neq Q_D) \wedge (Q_B \neq Q_C) \wedge (Q_B \neq Q_D) \wedge (Q_C \neq Q_D) \wedge$
$(|Q_A - Q_B| \neq 1) \wedge (|Q_A - Q_C| \neq 2) \wedge (|Q_A - Q_D| \neq 3) \wedge (|Q_B - Q_C| \neq 1) \wedge (|Q_B - Q_D| \neq 2) \wedge (|Q_C - Q_D| \neq 1)$

d) Assume a partial assignment is given, where Q-A is placed in row 3. Apply *backtracking search* starting from this partially-assigned CSP. Use the variable ordering (Q-B, Q-C, Q-D) and the variable domain order 1,2,3,4 to expand nodes in the search graph. List all variable assignment and removal operations, and any backtracking operations.

1. Check Q-B = 1 – **Assign Q-B to row 1**
2. Check Q-C = 1 – remove b/c conflict with Q-B row and Q-A diag
3. Check Q-C = 2 – remove b/c conflict with Q-B diag
4. Check Q-C = 3 – remove b/c conflict with Q-A row
5. Check Q-C = 4 – **Assign Q-C to row 4**
6. Check Q-D = 1 – remove b/c conflict with Q-B row
7. Check Q-D = 2 – **Assign Q-D to row 2**

e) Give a solution to this CSP.

Q-A=3
Q-B=1
Q-C=4
Q-D=2

Question 2.2

a) Construct a truth table to show that ¬(p ∨ q) is logically equivalent to (¬p ∧ ¬q).

Literals		Sentence	
p	q	¬(p ∨ q)	(¬p ∧ ¬q)
1	1	0	0
1	0	0	0
0	1	0	0
0	0	1	1

b) Given the premises (p ⇒ q) and (r ⇒ s), use the Propositional Resolution rule to prove the conclusion (p ∨ r ⇒ q ∨ s).

Drive out arrows from the premises:

(p ⇒ q) ⇔ (¬p ∨ q) (1)

(r ⇒ s) ⇔ (¬r ∨ s) (2)

Next we want the goal statements in CNF. First, drive out arrows from the goal statement:

(p ∨ r ⇒ q ∨ s) ⇔ ¬ (p ∨ r) ∨ (q ∨ s)

Then apply de Morgan's laws (twice) to drive in ANDs:

¬ (p ∨ r) ∨ (q ∨ s) ⇔ ¬((p ∨ r) ∧ ¬q ∧ ¬s))

Negate the goal (we are using proof by contradiction) and list the goal conjuncts separately:

(p ∨ r) (3)
¬q (4)
¬s (5)

Now apply resolution to the premises and goal statements:

Resolution of (1) and (4):

(¬p ∨ q)
¬q
¬p (6)

Resolution of (2) and (3):

(¬r ∨ s)
¬s
¬r (7)

Resolution of (3) and (6):

(p ∨ r)
¬p
r (8)

From (7) and (8) we have r ∧ ¬r, a contradiction. Thus we prove the original conclusion.

MODULE 3: DECISION MAKING UNDER UNCERTAINTY

Question 3.1

The *decomposability* axiom of the rational preferences utility model asserts that “there is no fun in gambling.” Explain how an agent whose preferences violate the decomposability axiom can be manipulated using the concept of a *money pump*.

Using a violation of the decomposability axiom, you can set up an intransitive preference loop and apply the money pump.

Your typical money pump is constructed for an intransitive loop of preferences: $A < B < C < A$. At each step of the money pump, a small payment is taken from the agent to swap to the next-preferred option.

Let's now assume we don't have C, but instead we have a lottery, $D = pA + (1-p)B$.

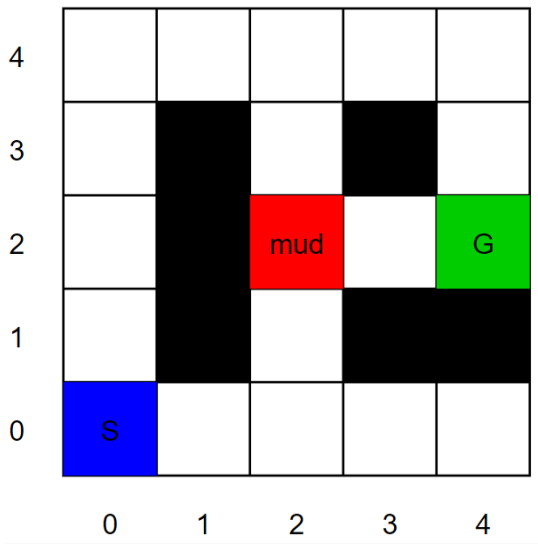
If an agent values the lottery element enough, that is, if it gets enough pleasure from gambling, we can imagine a situation where: $A < B < D$, and D realises one of A or B.

Let's cycle through these preferences to pump money out of the agent:

1. Assume the agent starts with A, and that you start with a stash of A and B. Repeat the following:
2. Trade B for A plus a small payment.
3. Offer the lottery D for B plus a small payment.
4. Realise D and give it to the agent.
5. If D realised A for the agent, go to 1; if D realised B for the agent, go to 2.

Question 3.2

Answer the following questions about the gridworld Markov decision process shown below:



The details are:

- The 5x5 grid has obstacles indicated by black tiles in three contiguous blocks, at coordinates (1,1), (1,2), (1,3); at (3,1) and (4,1); and at (3,3).
- The agent starts at the state labelled S, at coordinates (0,0).
- Its goal state is labelled G, at (4,2). **Assume the goal state is absorbing.**
- Entering the goal state earns the agent 10 points.
- Actions that cause collisions with the boundary or obstacles keep the agent at its current state (with no cost collision).
- The red square at (2,2) is mud, which can slow the agent down. The agent successfully moves out of the red square with probability 0.2, and remains stuck in the mud with probability 0.8.
- Assume $\gamma = 0.8$.

a) What is the transition function for each action starting in the mud? That is, write out *T(s,a,s')* for each *s'* with non-zero transition probability, starting at s=(2,2).

T((2,2),up,(2,2)) = 0.8
T((2,2),up,(2,3)) = 0.2

T((2,2),down,(2,2)) = 0.8
T((2,2),down,(2,1)) = 0.2

T((2,2),right,(2,2)) = 0.8
T((2,2),right,(3,2)) = 0.2

T((2,2),left,(2,2)) = 1

b) Initialise all values to 0, and compute three iterations of (synchronous) *value iteration* for the gridworld above. What are the value function iterates (i.e. approximate values) for each state after:

i. the first iteration,
Note V(4,2) = 0

V(3,2) = 10
V(4,3) = 10

ii. the second iteration, and
V(3,2) = 10 (same as above)
V(4,3) = 10 (same as above)
V(2,2) = 0.2*0.8*V(3,2) = 0.2*0.8*10= 1.6
V(4,4) = 0.8*V(4,3) = 0.8*10 = 8

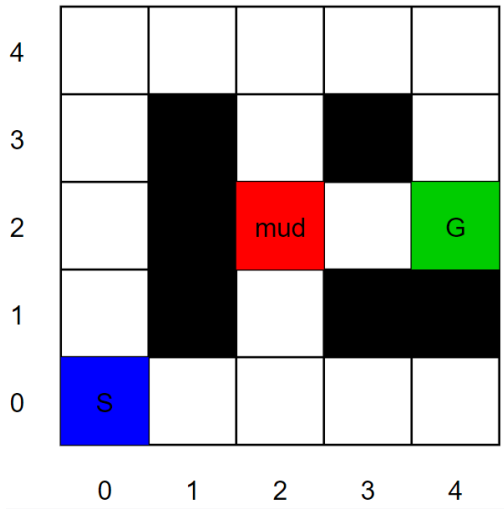
iii. the third iteration.
V(3,2) = 10 (same as above)
V(4,3) = 10 (same as above)
V(4,4) = 8 (same as above)
V(2,2) = 0.2*0.8*V(3,2) + 0.8*0.8*V(2,2) = 0.2*0.8*10 + 0.8*0.8*1.6 = 1.6 + 1.024
=2.624
V(2,1) = 0.8*V(2,2) = 0.8*1.6 = 1.28
V(2,3) = 0.8*V(2,2) = 0.8*1.6 = 1.28
V(3,4) = 0.8*0.8*10 = 6.4

All other iterates are equal to 0.

MODULE 4: LEARNING TO ACT

Question 4.1

Using the same gridworld as above, assume now you do not know anything about the transitions or rewards. The gridworld is shown again below:



You choose to use SARSA to solve this problem, and employ an epsilon-greedy exploration strategy with exploration probability $\epsilon = 0.2$ and exploration using uniform random sampling of any action otherwise.

Suppose you obtain the following observations:

s_t	a_t	s_{t+1}	reward	a_{t+1}
(2,1)	Up	(2,2)	0	Up
(2,2)	Up	(2,2)	0	Right
(2,2)	Right	(3,2)	0	Right
(3,2)	Right	(4,2)	10	Restart

a) What values does the Q-function attain if we initialise the Q-values to 0 and replay the experience in the table exactly two times? Use a learning rate $\alpha = 0.6$, and discount factor $\gamma = 0.8$. List only the non-zero approximate Q-values; all unstated Q-values are assumed to be equal to 0.

First experience replay:

- The first three actions result in no rewards and no non-zero values being propagated.
- The fourth observation, (s,a,s') = ((3,2),Right, (4,2)) returns r =10, and is followed by a' = restart.
- Based on this, the SARSA update for Q((3,2), Right) = 0+ 0.6(10 + 0.8*0 - 0) = 6.

Second experience replay:

- The first two actions result in no rewards and no non-zero values being propagated.
- The third observation, (s,a,s') = ((2,2),Right, (3,2)) returns r =0 and is followed by a' =Right. The next state (3,2), has nonzero Q-value Q((3,2), Right) = 6.

- Based on this, the SARSA update for Q((2,2), Right) = 0+ 0.6(0 + 0.8*6 - 0) = 2.88.
- The fourth observation, (s,a,s') = ((3,2),Right, (4,2)) returns r =10, and is followed by a' = restart. The sampled state-action pair ((3,2),Right) has nonzero Q-value Q((3,2), Right) = 6.
- Based on this, the SARSA update for Q((3,2), Right) = 6 + 0.6(10 + 0.8*0 - 6) = 8.4.

The final non-zero Q values are:
Q((2,2), Right) = 2.88
Q((3,2), Right) = 8.4

b) Using these Q-values and the epsilon-greedy exploration strategy describe above, what are the probabilities of taking each action next time the SARSA agent gets stuck in the mud?

By these Q-values, the greedy action is to move *Right* from the mud state, s = (2,2), and the exploration probability is 0.2.

Random sampling the 4 directions with uniform probability gives every action 0.05.

Taken together, we have the following action probabilities:

Right = 0.8 + 0.05 = 0.85
Up = 0.05
Left = 0.05
Down = 0.05.

MODULE 5: REASONING ABOUT OTHER AGENTS

Question 5.1

In the game of *Morra*, each player shows either one or two fingers and announces a number between 2 and 4. If a player's number is equal to the sum of the number of fingers shown, then her opponent must pay her that many dollars. The payoff is the net transfer, so that both players earn zero if both or neither guess the correct number of fingers shown.

In this game, each player has 6 strategies: she may show one finger and guess 2; she may show one finger and guess 3; she may show one finger and guess 4; or she may show two fingers and guess one of the three numbers.

a) There are two *weakly dominated strategies* in Morra. What are they?

It never pays to put out one finger and guess that the total number of fingers will be 4, because the other player cannot put out more than two fingers.

Likewise, it never pays to put out two fingers and guess that the sum will be 2, because the other player must put down at least one finger.

Remaining payoffs to Player A (row player)

	one 2	one 3	two 3	two 4
one 2	0	2	-3	0
one 3	-2	0	0	3
two 3	3	0	0	-4
two 4	0	-3	4	0

b) Imagine that player A can read player B's mind and guess how he plays before he makes his move. What *pure strategy* should player B use?

Use minimax reasoning. Minimise the greatest gain his opponent gains, which is equivalent to maximising the minimum it is guaranteed to receive.

Look the table above: Answer is (one 3)

c) Player B consults a textbook and decides to use randomisation to improve his performance in Morra. Ideally, if he can find a best mixed strategy to play, what would be his expected payoff?

Trick question - it depends what his opponent does!

Will his opponent adapt? What happens if Player B know that Player A favours one strategy over another? What happens if Player B know that Player A randomises uniformly over all strategies? …

Let's take the worst case and assume that Player A can adapt to whatever lottery Player B chooses and can itself randomise (i.e. pick a mixed strategy). If Player A can still read his mind to see his mixed strategy, player B should choose to make Player A indifferent between pure strategies. Given this:

1. Because the game is symmetric, there is a symmetric mixed strategy Nash equilibrium.

2. Because the game is zero-sum, the symmetric mixed strategy Nash equilibrium must have payoffs to both players of 0.

d) One possible mixed strategy is to play show one finger and call “3” with probability 0.6, and to show two fingers and call “3” with probability 0.4 (and play the other strategies with probability 0). Is this a Nash equilibrium strategy? Assume that Player B is risk neutral with respect to the game payoffs.

We know that in the symmetric Nash equilibrium of a zero-sum game, the payoff is zero.

We also know that both players are indifferent to the pure strategies in the support set of their equilibrium mixed strategy. If they are indifferent, and they equal zero in total, they should be 0 individually too. If one is greater than the others, then Player A would not mix, they would exploit this.

To check, see if A’s expected rewards for each pure action are zero. If they are, then Player A will have been induced to randomise by B’s mixed strategy– this would be consistent with a Nash equilibrium.

Is this the case?

sigma_B = {P_B(one 3) : 0.6, P_B(two 3) : 0.4}

R_A(one 2, sigma_B) = 0.6*2 + 0.4*(-3) = 1.2 - 1.2 = 0.0
R_A(one 3, sigma_B) = 0.6*0 + 0.4*0 = 0
R_A(two 3, sigma_B) = 0.6*0 + 0.4*0 = 0
R_A(two 4, sigma_B) = 0.6*(-3) + 0.4*4 = -1.8 + 1.6 = -0.2

So sigma_B is not consistent with a Nash equilibrium.