Planning and Search

Mock exam answers 1-3

Reminder of exam format

4 questions out of 6

Two hours

Question 1

1.

(a) Explain A^* search algorithm (give pseudocode) (10 marks)

(b) What is an admissible heuristic? Give an example.(5 marks)

(c) What is a consistent heuristic? Give an example. (5 marks)

(d) Give an example of a search problem and an admissible heuristic where a graph search using A^* is not optimal (does not return the best solution). (5 marks)

Question 1a

Explain A^* search algorithm (give pseudocode) (10 marks)

A^* uses a heuristic function h and expands the nodes in the order of their evaluation function (best first):

Evaluation function $f(n) = g(n) + h(n)$

$g(n) =$ cost so far to reach n

$h(n) =$ estimated cost to goal from n

$f(n) =$ estimated total cost of path through n to goal

Question 1a

Pseudocode for the tree search A^* might look like this:

```
function TREE-SEARCH-A*( problem, fringe) returns a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
/* maintain the fringe in the increasing order of f(n) */
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE(node)) then return node
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

Question 1a

For the graph search A^*, we also check if we have already seen the same state before:

```
function GRAPH-SEARCH-A*( problem, fringe) returns a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
/* maintain the fringe in the increasing order of f(n) */
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then return node
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe ← INSERTALL(EXPAND(node, problem), fringe)
    end
```

Question 1b

(b) What is an admissible heuristic? Give an example.(5 marks)

Answer. An admissible heuristic is one that never overestimates the true cost of reaching the goal.

Typical example is a straight line distance for path finding problems.

(Not required by the question but just for interest: admissible but not consistent heuristic. Suppose we get a list of railway standard ticket prices between the cities and use it as a heuristic to estimate distances (for example, assuming that 1 kilometer costs 100 pounds - very expensive to make it admissible, giving us very few kilometers between cities).

Question 1c

(c) What is a consistent heuristic? Give an example. (5 marks)

Answer. A heuristic is consistent if the estimate at the current node is at most the cost of the next step plus the estimate from there: $h(n) \leq cost(n, n') + h(n')$.

Question 1d

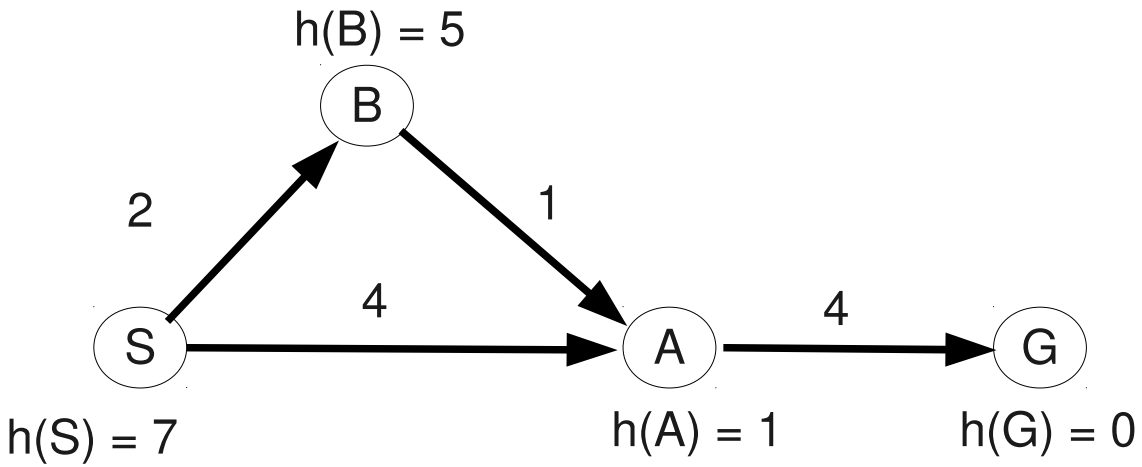
Give an example of a search problem and an admissible heuristic where a graph search using A^* is not optimal (does not return the best solution). (5 marks)

Answer.$h(S) = 7$, $cost(S, B) = 2$, $cost(S, A) = 4$, $h(B) = 5$, $h(A) = 1$, $cost(B, A) = 1$, $cost(A, G) = 4$, $h(G) = 0$. $f(A) = 5$ and $f(B) = 7$.

$$7 = h(S) \not\leq cost(S, A) + h(A) = 4 + 1$$

$$5 = h(B) \not\leq cost(B, A) + h(A) = 1 + 1$$

A will be expanded first, and the path through B which is cheaper will be discarded because A will be already in the closed list.


h(B) = 5
B
2
1
4
4
S
A
G
h(S) = 7
h(A) = 1
h(G) = 0

Question 2

(a) Explain the difference between state-space search algorithms and local search. What is the main reason for using local search? (5 marks)

(b)Explain hill climbing search (give pseudocode) (10 marks)

(c) Assume that the local search problem is to produce a natural number which is greater than 41 and is divisible by 5. The only way to generate successors is to add 1 or subtract 1 from the current state. Design an objective function for this problem. (5 marks)

(d) Explain the problem of 'getting trapped in local maxima' on the example from part (b) (an example of a 'local plateau' would do as well). How does stochastic hill climbing attempts to solve this problem? (5 marks)

Question 2a

Explain the difference between state-space search algorithms and local search. What is the main reason for using local search? (5 marks)

Answer. State-space search produces an expanding tree (or some other growing collection) of states, while local search manipulates a single state or a fixed number of states; the advantage of the latter is that it requires a lot less (and fixed size) space.

Question 2b

Explain hill climbing search (give pseudocode) (10 marks)

Answer. In the real exam you will have a choice of using an English description.

```
function HILL-CLIMBING( problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest-valued successor of current
        if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
        current ← neighbor
    end
```

Question 2c

Assume that the local search problem is to produce a natural number which is greater than 41 and is divisible by 5. The only way to generate successors is to add 1 or subtract 1 from the current state. Design an objective function for this problem. (5 marks)

Answer. The one I came up with is to take into account the difference between 41 and the current solution if it is smaller than 41 ($max(0, 41-x)$), and the remainer of division by 5 ($x \; mod \; 5$). The smaller these numbers, the better the solution. So the function is $f(x) = -(max(0, 41-x)+x \; mod \; 5)$. Must be possible to do better, but this kind of thing will get you full marks.

Question 2d

Explain the problem of 'getting trapped in local maxima' on the example from part (b) (an example of a 'local plateau' would do as well). How does stochastic hill climbing attempts to solve this problem? (5 marks)

Answer. The problem is when the solution is not found, but all successors of the current solution are worse (local maximum) or no better (local plateau). So the algorithm terminates without finding the solution. Stochastic hill climbing allows selecting a successor at random, which may help escape local maxima. With the function above, consider:
$f(20) = -(21 + 0) = -21$
$f(21) = -(20 + 1) = -21$
$f(22) = -(19 + 2) = -21$
so when $x = 21$, both successors ($x + 1$ and $x - 1$) have the same value of the objective function: x is on a local plateau.

Question 3

3 (a) Express the following problem in propositional logic:

> I want to invite some of the following people to a party: Alice, Ben, Chris and Dave. If I invite Alice, I would also have to invite Ben. I cannot invite Ben and Chris to the same party. I want to invite at least three of them (this condition you also need to express as a logical formula). (7 marks)

(b) Rewrite the formulas above in CNF (hint: the last condition, about inviting at least three people, is a bit of a pain to rewrite to CNF if you express it in the obvious way to begin with. It may be easier to work with an equivalent condition, 'I do not want to exclude any two of them from the party'). (8 marks)

(c) Trace DPLL algorithm on the resulting set of clauses. Point out pure symbols and unit clauses at each iteration. (10 marks)

Question 3a

I want to invite some of the following people to a party: Alice, Ben, Chris and Dave. If I invite Alice, I would also have to invite Ben. I cannot invite Ben and Chris to the same party. I want to invite at least three of them (this condition you also need to express as a logical formula). (7 marks)

Answer. Let A stand for inviting Alice and so on.

S1 $A \Rightarrow B$ (if invite Alice, then also invite Ben)

S2 $\neg(B \wedge C)$ (cannot invite Ben and Chris at the same time)

At least three of them: we can write this as

$$(A \wedge B \wedge C) \vee (A \wedge B \wedge D) \vee (A \wedge C \wedge D) \vee (B \wedge C \wedge D)$$

which is more natural.

However, it is also possible to say that we do not any two of them to be uninvited:

S3

$$\neg(\neg A \wedge \neg B) \wedge \neg(\neg A \wedge \neg C) \wedge \neg(\neg A \wedge \neg D) \wedge \neg(\neg B \wedge \neg C) \wedge \neg(\neg B \wedge \neg D) \wedge \neg(\neg C \wedge \neg D)$$

Question 3b

Rewrite the formulas above in CNF (hint: the last condition, about inviting at least three people, is a bit of a pain to rewrite to CNF if you express it in the obvious way to begin with. It may be easier to work with an equivalent condition, 'I do not want to exclude any two of them from the party'). (8 marks)

C1 $\neg A \vee B$ clause $[\neg A, B]$

C2 $\neg B \vee \neg C$ clause $[\neg B, \neg C]$

C3 $\neg(\neg A \wedge \neg B) = A \vee B$ clause $[A, B]$

similarly, a clause for every pair from A, B, C, D: $[A, B]$, $[A, C]$, $[A, D]$, $[B, C]$, $[B, D]$, $[C, D]$

Question 3c

Trace DPLL algorithm on the resulting set of clauses. Point out pure symbols and unit clauses at each iteration. (10 marks)

Answer.

Pure symbol heuristic: pure symbol is a symbol which occurs in all clauses with the same sign (for example only as $\neg A$). If a sentence has a model, then it has a model where pure symbols are assigned so as to make their literals true (for example A is assigned false). This is the value which this heuristic assigns to the symbol. Purity is recalculated as some clauses become true and can be ignored (so purity is defined relative to the set of remaining clauses)

Unit clause heuristic: unit clause is a clause with only one literal. In DPLL, it is a clause where only one symbol is yet unassigned. Unit clauses also have obvious assignment (for example for $\{\neg A\}$ to be true we have to assign false to A. This is the value which this heuristic assigns to the symbol.

function DPLL-SATISFIABLE?(s) **returns** *true* or *false*

inputs: s, a sentence in propositional logic

$clauses \leftarrow$ the set of clauses in the CNF representation of s

$symbols \leftarrow$ a list of the proposition symbols in s

return DPLL($clauses, symbols, [\,]$)

---

function DPLL($clauses, symbols, model$) **returns** *true* or *false*

if every clause in $clauses$ is true in $model$ **then return** *true*

if some clause in $clauses$ is false in $model$ **then return** *false*

$P, value \leftarrow$ FIND-PURE-SYMBOL($symbols, clauses, model$)

if P is non-null **then return** DPLL($clauses, symbols$–$P, [P = value|model]$)

$P, value \leftarrow$ FIND-UNIT-CLAUSE($clauses, model$)

if P is non-null **then return** DPLL($clauses, symbols$–$P, [P = value|model]$)

$P \leftarrow$ FIRST($symbols$); $rest \leftarrow$ REST($symbols$)

return DPLL($clauses, rest, [P = true|model]$) **or** DPLL($clauses, rest, [P = false|model]$)

Question 3c

Initial clauses: $[\neg A, B]$, $[\neg B, \neg C]$, $[A, B]$, $[A, C]$, $[A, D]$, $[B, C]$, $[B, D]$, $[C, D]$

Initial symbols: A, B, C, D

Pure symbol: D. Assign true to D. The clauses where D occurs also become true, so we skip them.

Next clauses: $[\neg A, B]$, $[\neg B, \neg C]$, $[A, B]$, $[A, C]$, $[B, C]$. Model: $D = true$

Next symbols: A, B, C

No pure symbol, no unit clause. First symbol is A. Need to call DPLL for $A = true, D = true$ and $A = false, D = true$. Start with the first one ($A = true$).

Skip the clauses where A occurs because they are true, and remove $\neg A$ from clauses where it occurs (because it is false).

Next clauses: $[B]$, $[\neg B, \neg C]$, $[B, C]$. Model: $A = true, D = true$

Next symbols: B, C

No pure symbols.

Unit clause: $[B]$. Set B to true. Same as before: remove clauses where B occurs positively, and remove $\neg B$ from clauses.

Next clauses: $[\neg C]$. Model: $B = true, A = true, D = true$.

Unit clause: $[\neg C]$. Assign false to C. Model: $C = true, B = true, A = true, D = true$.

Return true (all clauses true).