Answers G52PAS 2013-2014

BOOKWORK VS PROBLEM SOLVING: questions are problem solving unless stated otherwise. Students have done exercises of the same type (e.g. producing a trace of an algorithm) but sufficiently different, so that memorising a solution to the exercise is not useful.

1. This question is on classical search (25 marks in total).

   (a) Figure 1 gives a graphic representation of bus stations $\{a, b, c, d, e, f, g, h, i, j, k, l, m\}$ connected by bus routes. The actual length of the route connecting any two stations is given in km, e.g., the actual distance between a and b is 240 km. Consider the problem of finding a route from a to m. Construct two search trees generated as the result of: (i) using greedy search and (ii) A* search for solving this problem. Label the nodes expanded for both trees with the appropriate costs. For both trees, you should use the straight-line distance $v(x)$ between the station x and m: $v(a) = 800$, $v(b) = 600$, $v(c) = 595$, $v(d) = 610$, $v(e) = 400$, $v(f) = 400$, $v(g) = 420$, $v(h) = 320$, $v(i) = 315$, $v(j) = 100$, $v(k) = 250$, $v(l) = 50$, and $v(m) = 0$. Which search algorithm gives a lower total distance travelled? (15 marks)

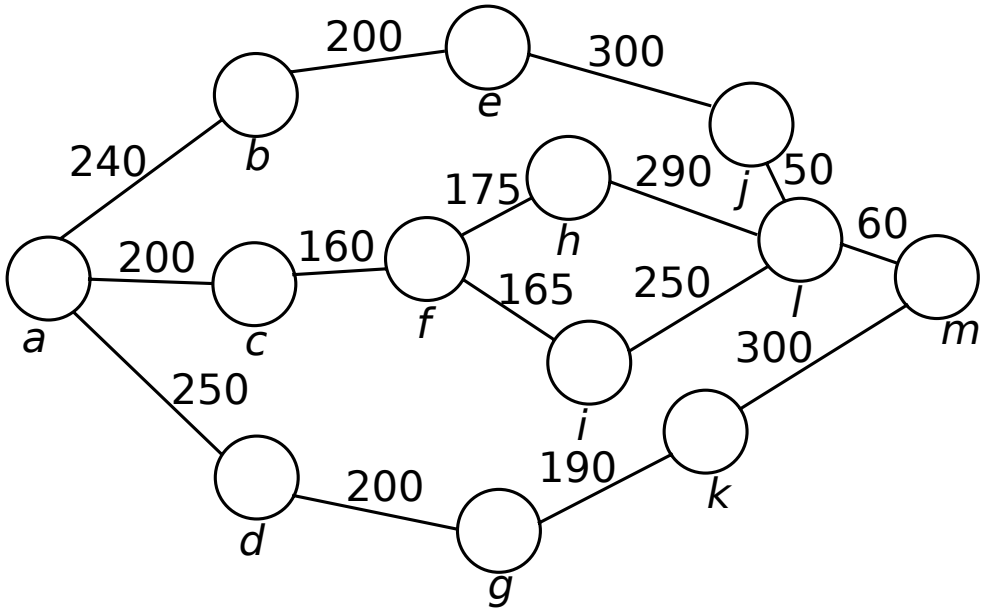

Figure 1:

Answer. Figures 2 and 3 give the trees generated as a result of greedy search and A* search, respectively. Greedy search returns the path $a-c-f-i-l-m$ with a total distance of $200+160+165+250+60 = 835$ km. A* search returns the path $a-c-f-i-l-m$ with a total distance of $200+160+165+250+60 = 835$ km. Both returns the same total distance travelled.

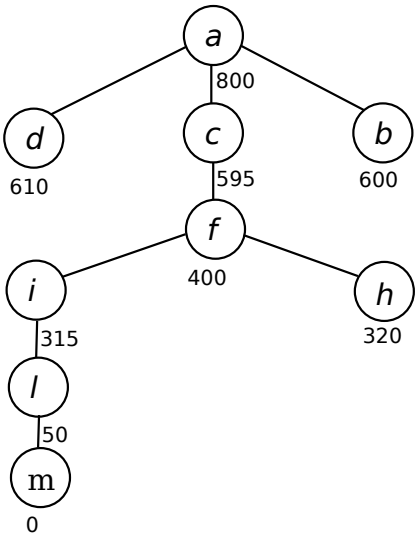

Figure 2: Greedy Search

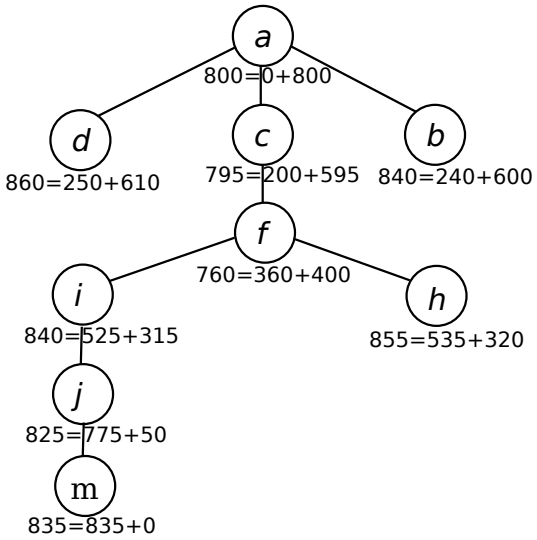

Figure 3: A* Search

(b) Explain how hill-climbing works (give algorithm in pseudocode). What is the problem typically associated with hill-climbing? What are the methods to address this problem? (10 marks)

Answer. (BOOKWORK) A hill-climbing algorithm starts with an initial state and then iteratively generates successor states and selects the state with the highest objective value. It terminates if it cannot improve on the current state.

```
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
```

Hill Climbing is typically associated with the problem of getting stuck in local optima. There are three methods that can address this problem: (i) random restart, (ii) random sideways moves (jumps), (iii) tabu list.

2. This question is on search with non-deterministic actions and partial observability (25 marks in total).

   (a) Consider the following and-or tree representing a search problem with non-deterministic actions (actions are indicated next to the arrows, and the states $s4$ and $s5$ satisfy the goal test). Give a subtree which represents a solution to the problem, and explain why it is a solution (state which properties a solution should satisfy). (5 marks)

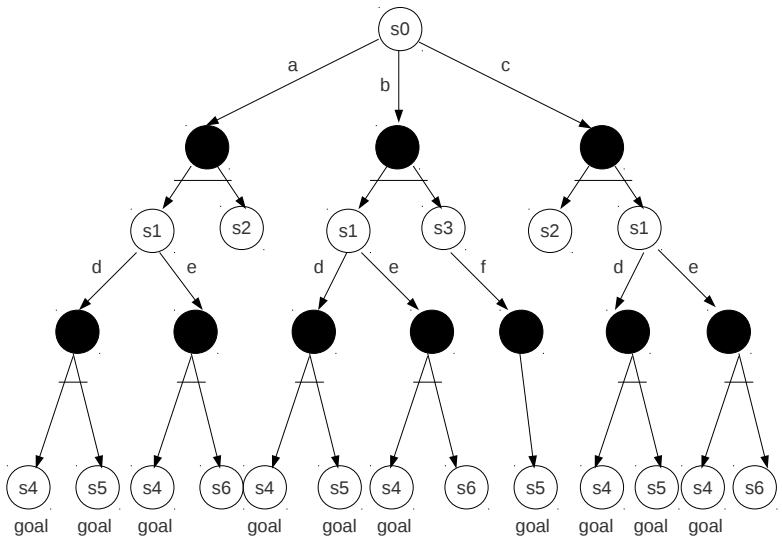

Figure 4:

Answer. (Definition of a solution is bookwork).
A solution to an and-or search problem is a subtree of the and-or tree which contains one outgoing edge for each or-node and all outgoing edges for each and-node and where all leaves are goal states.

   (b) Give the corresponding conditional plan. (4 marks)
   Answer. $b; if\, s1\, d; else\, f$.

   (c) Consider a system with four physical states: an agent can be either indoors or outdoors, and outdoors can be either cold or warm. More precisely,

   in state s_0 the agent is indoors and outdoors is cold
   in s_1 the agent is indoors and outdoors is warm
   in s_2 the agent is outdoors and outdoors is cold
   in s_3 the agent is outdoors and outdoors is warm.

   The environment is only partially observable. The agent can always perceive whether it is indoors or outdoors. However it can only perceive whether outdoors is cold or warm when it is outdoors. Possible percepts are (combinations of) IN, OUT, $COLD$, $WARM$. The agent cannot change the temperature outdoors, but it has

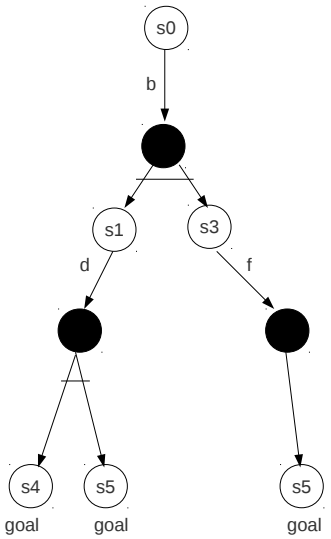

Figure 5:

actions *in* and *out* to change its location. If the agent is indoors, executing *out* brings it outdoors, and if it is outdoors, executing *in* brings it indoors. For example, $Result(s_0, out) = s_2$.

i. What is the value of $Result(s_3, in)$? (2 marks)
Answer. $Result(s_3, in) = s_1$.

ii. What is the value of $Percept(s_1)$? (2 marks)
Answer. $Percept(s_1) = \{IN\}$

iii. What is the value of $Percept(s_2)$? (2 marks)
Answer. $Percept(s_2) = \{OUT, COLD\}$

iv. Suppose the agent knows that it is indoors and does not know whether it is cold or warm outdoors. What is its belief state b_0 (what are the physical states it considers possible)? (2 marks)
Answer. $b_0 = \{s_0, s_1\}$

v. What is $\widehat{b_0} = Predict(b_0, out)$ (all physical states the agent may be in after executing *out* in b_0)? (2 marks)
Answer. $\widehat{b_0} = \{s_2, s_3\}$

vi. What are the possible percepts in $\widehat{b_0}$? (2 marks)
Answer. In s_2, the percept is $\{OUT, COLD\}$, and in s_3, $\{OUT, WARM\}$.

vii. For each percept o which is possible in $\widehat{b_0}$, state what is the value of $Update(\widehat{b_0}, o)$ (which physical states remain in $\widehat{b_0}$ if the percept is o). (2 marks)
Answer.
$Update(\widehat{b_0}, \{OUT, COLD\}) = \{s_2\}$
$Update(\widehat{b_0}, \{OUT, WARM\}) = \{s_3\}$

viii. What is the value of $Results(b_0, out)$ (what are all the belief states the agent may have after executing *out* and perceiving the environment)? (2 marks)

Answer.
$Results(b_0, out) = \{\{s_2\}, \{s_3\}\}$

3. This question is on planning in general and its relation to search (25 marks in total).

   (a) Explain the difference between how a *classical search problem* and a *classical planning problem* are formulated, and between solutions to a search problem and a planning problem. (5 marks)

   Answer. BOOKWORK. A search problem is formulated in terms of (1) a set of states, (2) for each state, a list of actions applicable in this state, (3) for each pair of a state and applicable action, the resulting state, (4) initial state(s), (5) goal state(s) (or goal test). A planning problem is formulated in terms of (1) action schemas with preconditions and effects, (2) initial state(s) description in terms of fluents, (3) goal state description in terms of fluents. Planning problems use factored representations of states, while search problems consider states as simple atomic entities. A solution to a search problem is a totally ordered sequence of actions. A solution to a planning problem may be a partially ordered set of actions.

   (b) Define the following planning problem in Planning Domain Description language (specify predicates, objects, initial state, goal specification, action schemas). (10 marks)

   > There are three locations, $L1$, $L2$, and $L3$, and two boxes, $B1$ and $B2$. A robot can push a box x from location y to location z, provided the box and the robot are at y and y is different from z. A robot can move from x to y if it is at x and x is different from y.
   > The actions are: $Push(x, y, z)$ (push x from y to z) and $Move(x, y)$ (move from x to y).
   > In the initial state, $B1$ is at $L1$ and $B2$ is at $L2$, and the robot is at $L3$. The goal is to have all boxes at $L3$.

   Answer.

   Predicates are: $At(x, y)$ (object x is at location y)

   Objects: $B1, B2, L1, L2, L3, R$ (R is the robot)

   Initial state: $At(B1, L1), At(B2, L2), At(R, B3)$

   Goal specification: $At(B1, L3), At(B2, L3)$.

   Action schemas:

   $Push(x, y, z)$
   Precondition: $At(x, y), At(R, y), \neg(x = y)$
   Effect: $At(x, z), At(R, z), \neg At(x, y), \neg At(R, y)$

   $Move(x, y), \neg(x = y)$
   Precondition: $At(R, x)$
   Effect: $At(R, y)$, $\neg At(R, x)$

   (c) How do you turn the planning domain from part (b) into a search problem? Say what the states would be (no need to list them all, just give one example), how to determine whether an action is applicable in a state, and what the result of executing an action in a state would be. (5 marks)

   Answer. States are sets of ground fluents, for example the initial state s_0 is $\{At(B1, L1)$, $At(B2, L2)$, $At(R, L3)\}$ (all other possible ground facts are assumed false). An action $a \in Actions(s)$ if $s \models Precond(a)$. $Results(s, a) = (s \setminus Del(a)) \cup Add(a)$, where $Del(a)$ is the list of literals which appear negatively in the effect of a and $Add(a)$ is

the list of positive literals in the effect of a. For example, $Result(s_0, Move(L3, L1)) = \{At(B1, L1), At(B2, L2), At(R, L1)\}$.

(d) What is an admissible heuristic? State which admissible heuristic could be used for performing forward (progression) search for the problem from part (b) (assuming the cost of each action is 1). (5 marks)

Answer. An admissible heuristic never overestimates the cost of a solution. A possible admissible heuristic $h(n)$ would return the number of boxes which are not in $L3$ in the state of n. Since at least one action is required to transfer the box to $L3$, this heuristic will not overestimate the cost of reaching the goal.

4. This question is on various planning algorithms and on situation calculus (25 marks in total).

   Consider the following planning domain (a version of the monkey and banana puzzle from Russell and Norvig). There are two boxes $B1$ and $B2$. The agent can stack box x on top of another box y if both x and y are clear (have nothing on top). The agent can unstack boxes. When there is a stack of two unpainted boxes, the agent can climb on top of them and get a banana. The agent can also paint a box if it is clear. After the box is painted, it cannot be used for climbing.

   The actions are:

   $Stack(x, y)$:
   PRECOND: $Clear(x) \wedge Clear(y) \wedge \neg Painted(x) \wedge \neg Painted(y) \wedge \neg(x = y)$
   EFFECT: $On(x, y) \wedge \neg Clear(y)$

   $Unstack(x, y)$:
   PRECOND: $On(x, y) \wedge Clear(x)$
   EFFECT: $\neg On(x, y) \wedge Clear(y)$

   $Paint(x)$:
   PRECOND: $Clear(x)$
   EFFECT: $Painted(x)$

   $GetBanana(x, y)$:
   PRECOND: $On(x, y) \wedge \neg Painted(x)$
   EFFECT: $Have(Banana)$

   In the initial state, all the boxes are clear, none of them are painted, nothing is on top of another box and the agent does not have the banana:

   Initial state: $Clear(B1)$, $Clear(B2)$.

   Goal: $Painted(B1), Have(Banana)$.

   (a) Solve this problem using partial order planning; trace the search from the initial empty plan to a complete solution, explaining each step. (15 marks)

   Answer.

   $Steps = \{Start, Finish\}$ where the effect of $Start$ is $\{Clear(B1), Clear(B2)\}$, and the $Finish$ step has $\{Painted(B1), Have(Banana)\}$ as its precondition.

   $Links = \{\ \}$

   $Orderings = \{Start \prec Finish\}$

   Open conditions: $Painted(B1)$, $Have(Banana)$ (for the $Finish$ step).

   Suppose we choose the first open condition as a selected subgoal.

   An action which would make it true is $Paint(B1)$. We add it as a step to the partial plan, and a link from it to $Finish$:

   $Steps = \{Start, Finish, Paint(B1)\}$

$Links = \{\ Paint(B1) \overset{Painted(B1)}{\longrightarrow} Finish\}$

$Orderings = \{Start \prec Finish, Start \prec Paint(B1),\ Paint(B1) \prec Finish\}$

Open conditions: $Have(Banana)$ (for the $Finish$ step).

An action which makes it true is $GetBanana(x, y)$. We add it as a step in in the partial plan, and add a link from it to $Finish$. We also use it with a substitution for simplicity:

$Steps = \{Start, Finish, Paint(B1), GetBanana(B1, B2)\}$

$Links = \{\ Paint(B1) \overset{Painted(B1)}{\longrightarrow} Finish, GetBanana(B1, B2) \overset{Have(Banana)}{\longrightarrow} Finish\}$

$Orderings = \{Start \prec Finish, Start \prec Paint(B1), Paint(B1) \prec Finish, Start \prec GetBanana(x, y), GetBanana(B1, B2) \prec Finish\}$

Open conditions: $On(B1, B2), \neg Painted(B1)$ (for the $GetBanana(B1, B2)$ step).

Let us take the first open condition. It can be achieved by the action $Stack(B1, B2)$.

$Steps = \{Start, Finish, Paint(B1), GetBanana(B1, B2), Stack(B1, B2)\}$

$Links = \{\ Paint(B1) \overset{Painted(B1)}{\longrightarrow} Finish, GetBanana(B1, B2) \overset{Have(Banana)}{\longrightarrow} Finish,$
$Stack(B1, B2) \overset{On(B1,B2)}{\longrightarrow} GetBanana(B1, B2)\}$

Orderings same as before with $Start \prec Stack(B1, B2)$ and $Stack(B1, B2) \prec Finish$ added, and also because $Stack(B1, B2)$ makes true the condition for $GetBanana(B1, B2)$, $Stack(B1, B2) \prec GetBanana(B1, B2)$.

Open conditions $\neg Painted(B1), Clear(B1), Clear(B2), \neg Painted(B1), \neg Painted(B2)$, $\neg(B1 = B2)$. All of them are made true by $Start$. Links add

$Start \overset{\neg Painted(B1)}{\longrightarrow} GetBanana(B1, B2)$, $Start \overset{Clear(B1)}{\longrightarrow} Stack(B1, B2)$, and so on, including $Start \overset{\neg Painted(B1)}{\longrightarrow} Stack(B1, B2)$.

Now we see that $Paint(B1)$ destroys the condition achieved by $Start$ and necessary for both $Stack(B1, B2)$ and $GetBanana(B1, B2)$. We cannot promote $Paint(B1)$ to before $Start$, so we demote it to after $GetBanana(B1, B2)$ instead. The resulting ordering is strict:

$Start \prec Stack(B1, B2) \prec GetBanana(B1, B2) \prec Paint(B1) \prec Finish$.

(b) Which problem with the goal stack planning algorithm is addressed by the partial order planning algorithm? (4 marks)

Answer. (BOOKWORK)

The goal stack algorithm commits to solving sub-goals in a particular order, which may be wrong (achieving one subgoal first may make it impossible to achieve another subgoal). For example, painting the box first will make it impossible to use it for climbing to get the banana. Partial order planning does not commit to any order on subgoals apart from imposing ordering constraints when necessary to avoid clobbering.

(c) Write successor state axioms in situation calculus for fluents $On(x, y, s)$ and $Painted(x, s)$ assuming action definitions above. (6 marks)

Answer.

$\forall x \forall y \forall a \forall s(On(x, y, Result(a, s)) \Leftrightarrow ((a = Stack(x, y)) \vee (On(x, y, s) \wedge \neg(a = Unstack(x, y)))))$

$\forall x \forall a \forall s(Painted(x, Result(a, s)) \Leftrightarrow (a = Paint(x) \vee Painted(x, s)))$