Answers G52PAS 2012-2013

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 tree search.

   (a) Describe briefly the basic idea of tree search algorithms. Explain how a general tree search algorithm works. Your answer should explain what are the nodes and branches in a search tree and mention node expansion and search strategy. (6 marks)

   Answer. BOOKWORK. Tree search algorithms explore the state space in an offline manner.

   Tree search algorithms construct a search tree where the root node corresponds to the initial state, branches are actions and nodes contain states in the state space of the problem. Tree search algorithms maintain a data structure called an open list (or frontier, or fringe) of leaf nodes to expand. Expanding a node means checking whether it contains the goal state (in which case the algorithm terminates), otherwise generating all of its successor nodes (one for each legal action applicable in the node's state) and adding them to the open list. The choice of the node for expansion is made according to some expansion or search strategy (for example, FIFO expansion strategy gives BFS).

   A general tree search algorithm works as follow:

   function TREE-SEARCH(*problem, strategy*) **returns** a solution, or failure
   initialize the search tree using the initial state of *problem*
   loop do
   if there are no candidates for expansion **then return** failure
   choose a leaf node for expansion according to *strategy*
   if the node contains a goal state **then return** the corresponding solution
   else expand the node and add the resulting nodes to the search tree
   end

   (b) Explain what it means for a search strategy to be complete. Give an example of a complete search strategy. (2 marks)

   Answer. BOOKWORK. A strategy is complete if it always finds a solution if it exists. Breadth-first search (for finite number of successors) is complete.

   (c) BOOKWORK. Explain what it means for a search strategy to be optimal. Give an example of an optimal search strategy. (2 marks)

   Answer. A strategy is optimal if it always returns the least-cost solution. A^* is optimal with admissible heuristics.

   (d) Figure 1 below gives the graphic representation of cities $\{a, b, c, d, e, f, g, h, i\}$ interconnected by highways in a country. The actual distance of the highway connecting two cities is given in km. For example, the highway for $\{a, b\}$ is 100 km in actual distance. Construct two trees generated as the result of: (i) using greedy search and (ii) A* search for the problem where the starting city is a and the ending city is i. Label the nodes expanded for both trees with the appropriate costs. For both trees, you should use the straight-line distance $v(n)$ between the city n and

i: $v(a) = 300$, $v(b) = 195$, $v(c) = 120$, $v(d) = 123$, $v(e) = 125$, $v(f) = 260$, $v(g) = 235$, $v(h) = 55$, and $v(i) = 0$. Which search algorithm gives a lower total distance travelled? (15 marks)

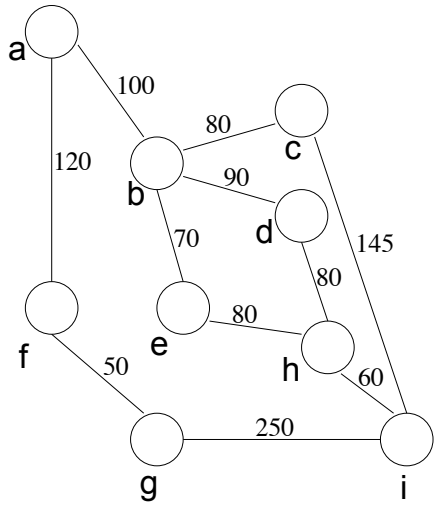


Figure 1: Cities

Answer. Figures 2 and 3 give the trees generated as a result of greedy search and A* search, respectively. Greedy search returns the path $a - b - c - i$ with a total distance of $100 + 80 + 145 = 325$ km. A* search returns the path $a - b - e - h - i$ with a total distance of $100 + 70 + 80 + 60 = 310$ km. A* search returns a lower total distance travelled.

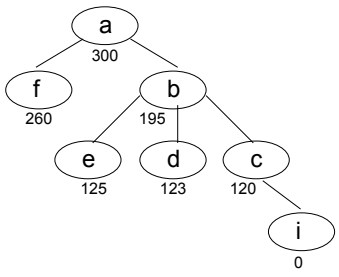


Figure 2: Greedy Search

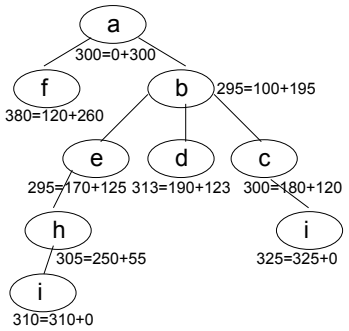


Figure 3: A* Search

2. This question is on local search.

   (a) Explain how hill-climbing works. (6 marks)

   Answer. BOOKWORK. A hill-climbing algorithm starts with an initial state and then iteratively generates successor states and select 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
```

   (b) Explain how genetic algorithms work. Your answer should include the terms chromosome, fitness function, crossover and mutation. (6 marks)

   Answer. BOOKWORK. Genetic algorithm is essentially stochastic local beam search which generates successors from *pairs* of states. It works with k states (chromosomes or individuals). This set is called population. Original population is randomly generated. Each individual represented as a string (chromosome). Each individual is rated by a maximising objective function (*fitness function*). Probability of being chosen for reproduction is directly proportional to fitness. Two parents produce offspring by *crossover*. Then with some small probablity, mutation is applied (bits of the string changed).

```
function GENETIC-ALGORITHM(population,FITNESS-FN) returns an individual
    inputs: problem, a set of individuals
            FITNESS-FN, a function that measures the fitness of an individual
    repeat
        new-population ← empty set
        for i=1 to SIZE(population) do
            x ← RANDOM-SELECTION(current,FITNESS-FN)
            y ← RANDOM-SELECTION(current,FITNESS-FN)
            child ← REPRODUCE(x, y)
            if (small random probability) then child ← MUTATE(child)
            add child to new-population
    until some individual is fit enough or enough time has elapsed
    return the best individual in population, according to FITNESS-FN
```

```
function REPRODUCE( x, y) returns an individual
    inputs: x, y, parent individuals
    n ← LENGTH(x); c ← random number from 1 to n
    return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c+1, n))
```

(c) Consider the following search problem:

- the set of states $S = \{s_0, s_1, s_2, s_3, s_4, s_5\}$
- successors of s_0 are $\{s_0, s_1, s_2\}$
- successors of s_1 are $\{s_1, s_2, s_3\}$
- successors of s_2 are $\{s_2, s_3\}$
- successors of s_3 are $\{s_0, s_3, s_4\}$
- successors of s_4 are $\{s_4, s_5\}$
- successors of s_5 are $\{s_2, s_3, s_5\}$
- objective function f is as follows: $f(s_0) = 0$, $f(s_1) = 3$, $f(s_2) = 2$, $f(s_3) = 4$, $f(s_4) = 1$, and $f(s_5) = 5$.

i. Which (if any) states are local maxima and global maxima? (3 marks)
Answer. s_3 is a local maximum, s_5 is a global maximum.

ii. Trace hill climbing search starting in state s_0 (indicate which state is considered at each iteration, and which solution is returned when the search terminates). Does it return the optimal solution? (5 marks)
Answer. The hill-climbing trace is given by:
Iteration 1: The highest-value successor of s_0 is s_1 ($f(s_1) = 3$).
Iteration 2: The highest-value successor of s_1 is s_3 ($f(s_3) = 4$).
Iteration 3: The highest-value successor of s_3 is s_3 ($f(s_3) = 4$).

Hill-climbing terminates in the third iteration and does not return the global maximum solution, only the local maximum.

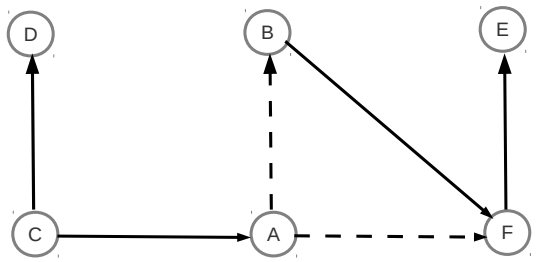
iii. Consider a stochastic hill-climber procedure for solving the same problem. In every iteration one successor state is randomly chosen from the set of successors. If the current state's objective function is strictly higher than that of the chosen successor, the current state is returned and the procedure terminates. If the random successor's objective function is the same or higher, than it is chosen as the successor. The random successor is chosen by drawing a random number r from $[0, 1]$. If the current state has n successors, listed in the order above, each of them gets assigned an interval, $[0, 1/n], (1/n, 2/n], \ldots, (n - 1/n, 1]$, and the successor state is chosen depending on which interval r belongs to. For example, for s_2 with two successors $\{s_2, s_3\}$, the intervals are $[0, 0.5]$ for s_2 and $(0.5, 0]$ for s_3, and if r is 0.3 then s_2 is chosen, and if r is 0.7 then s_3 is chosen.Trace this procedure starting in state s_0 for a sequence of r given by $(0.2, 0.4, 0.8, 0.6)$. Indicate which state is considered at each iteration, which successor state's interval r is in, and which solution is returned when the search terminates. Does it return the optimal solution? (5 marks)

Answer. The trace is given by:
Generation 1: The selected successor of s_0 is s_0 ($r = 0.2$ in s_0's interval $[0, 1/3]$, $f(s_0) = 0$).
Generation 2: The selected successor of s_0 is s_1 ($r = 0.4$ in s_1's interval $(1/3, 2/3]$, $f(s_1) = 3$).
Generation 3: The selected successor of s_1 is s_3 ($r = 0.8$ in s_3's interval $(2/3, 1]$, $f(s_3) = 4$).
Generation 4: The selected successor of s_3 is s_3 ($r = 0.6$ in s_2's interval $(1/3, 2/3]$, $f(s_3) = 4$).

It terminates in the fourth iteration and does not return the global maximum solution, only the local maximum.

3. This question is on search with non-deterministic actions.

   Consider the following search problem. The set of states corresponds to a set of locations on the map (the state A corresponds to being in location A, etc.). Actions correspond to following a directed edge on the map (going from one location to another). If there is an edge from x to y then it is possible to perform the action $go(x, y)$. All actions apart from two are deterministic and have the expected result, namely going from x to y results in being in y. The only exceptions are actions which involve moving out of A: the action of moving from A to F has two possible outcomes, one being in F and another being in B. Similarly, the action of moving from A to B also has two possible outcomes, one being in B and another being in F. The initial state is being in C and the goal is to reach E.



   (a) Give a list of all possible actions in state A and a list of all possible actions in state C. (2 marks)

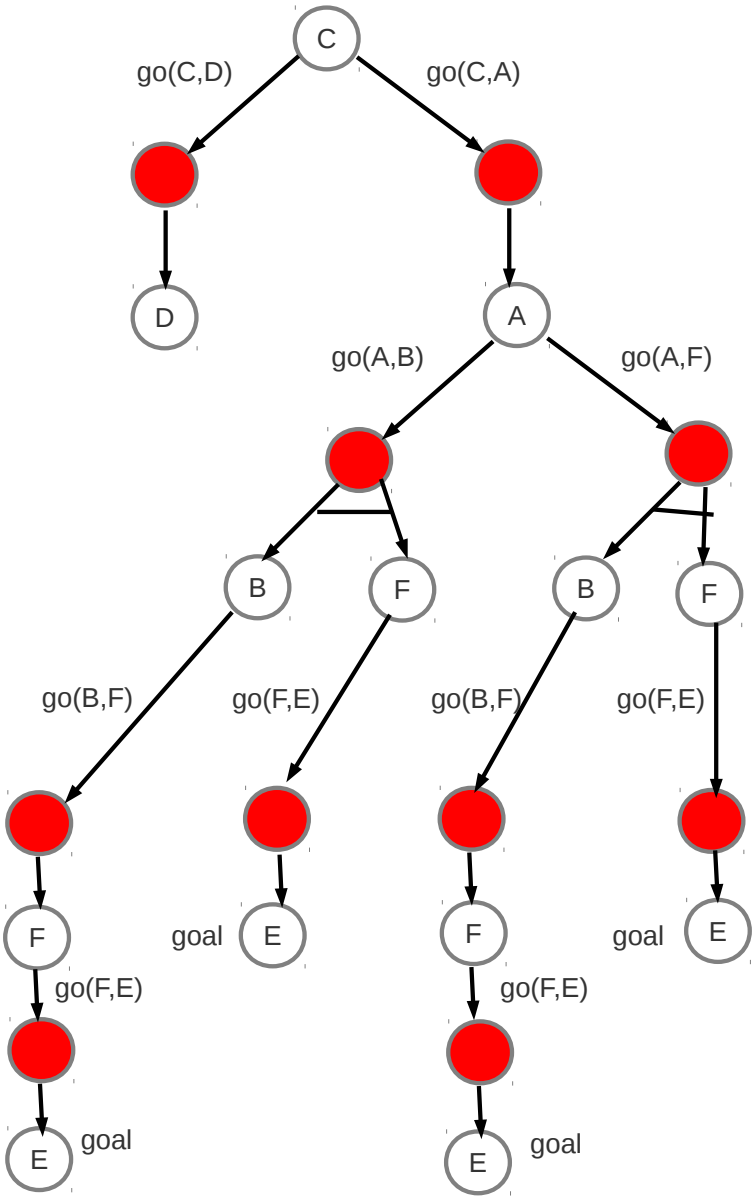
   Answer. (one point for each state)
   $Actions(A) = \{go(A, B), go(A, F)\}$
   $Actions(C) = \{go(C, A), go(C, D)\}$

   (b) For states A and C, and every action which is possible there, define the function $Results(state, action)$. (3 marks)

   Answer. (approx. 2 points for A, 1 point for C)
   $Results(A, go(A, B)) = \{B, F\}$
   $Results(A, go(A, F)) = \{B, F\}$
   $Results(C, go(C, A)) = \{A\}$
   $Results(C, go(C, D)) = \{D\}$

(c) Draw the and-or search tree for this search problem. (7 marks)

Answer.



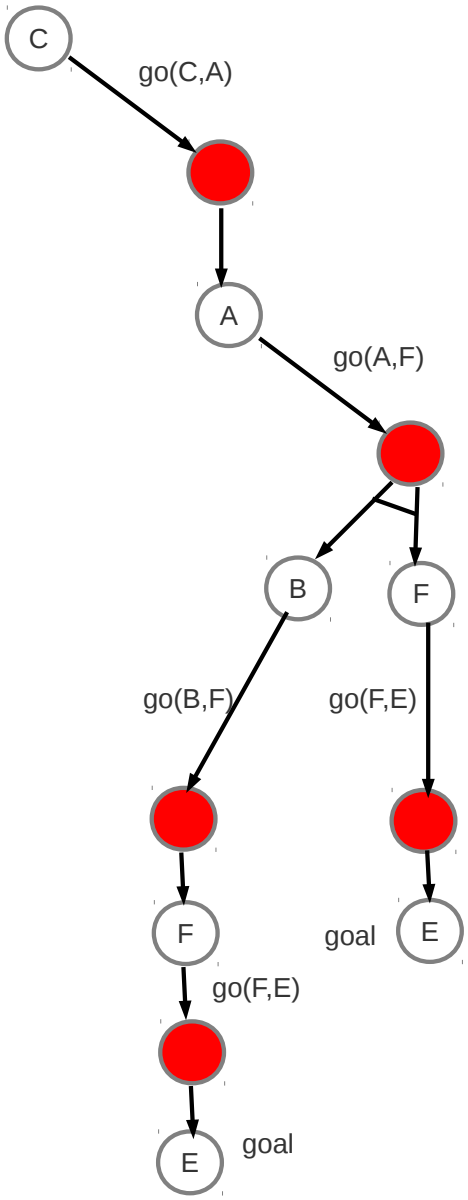
(d) Define what is a solution to an and-or search problem. (3 marks)

Answer. 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.

(e) Draw a subtree which is a solution to the problem. (5 marks)

Answer.



(f) State the corresponding conditional plan. (5 marks)

Answer.

$$\textbf{go}(\mathbf{C},\mathbf{A})\ \textbf{go}(\mathbf{A},\mathbf{F})\ \textbf{if B then}\ [\textbf{go}(\mathbf{B},\mathbf{F})\ \textbf{go}(\mathbf{F},\mathbf{E})]\ \textbf{else go}(\mathbf{F},\mathbf{E})$$

4. This question is on situation calculus.

   Consider the following problem. An agent which is a robot vacuum cleaner is moving between several rooms. It can perform the following actions:

   - $Suck(x)$ which means sucking up dirt in room x. This action is possible when the agent is in room x. The result of this action is that room x is clean.
   - $Move(x, y)$ which means move from room x to room y. This action is possible if the agent is in room x, and y is different from x. The result of this action is that the agent is no longer in room x, and is in room y.

   The fluents are: $In(x, s)$ which means that the agent is in room x in situation s, and $Clean(x, s)$ which means that room x is clean in situation s.

   (a) Write possibility axioms for the actions. (6 marks)

   Answer. (3 points for each)

   $$\forall x \forall s[Poss(Suck(x), s) \Leftrightarrow In(x, s)]$$

   $$\forall x \forall y \forall s[Poss(Move(x, y), s) \Leftrightarrow (In(x, s) \wedge x \neq y)]$$

   (b) Write successor state axioms for the fluents. (10 marks)

   Answer. (5 points for each)

   - $\forall x \forall a \forall s \forall y[In(x, Result(a, s)) \Leftrightarrow ((In(x, s) \wedge a \neq Move(x, y)) \vee (\neg In(x, s) \wedge In(y, s) \wedge a = Move(y, x)))]$
   - $\forall x \forall a \forall s[Clean(x, Result(a, s)) \Leftrightarrow ((Clean(x, s) \vee (\neg Clean(x, s) \wedge In(x, s) \wedge a = Suck(x)))]$

   (c) Suppose in S_0, the agent is in room A, and both rooms A and B are not clean. Find a substitution for the variables a_1, a_2, a_3 which makes the following sentence true: (5 marks)

   $$Clean(A, Result(a_3, Result(a_2, Result(a_1, S_0)))) \wedge$$

   $$Clean(B, Result(a_3, Result(a_2, Result(a_1, S_0))))$$

   Answer. The substitution is $a_1/Suck(A), a_2/Move(A, B), a_3/Suck(B)$.

   (d) Explain what the frame problem is and why successor state axioms can be considered to be a solution to it. (4 marks)

   Answer. BOOKWORK. The problem is specifying which properties of the world *do not* change as a result of an action, in a compact way. It is usually the case that there are many more actions which do not change the value of a fluent than there are actions that do. It is possible to write a frame axiom for every action and every fluent not changed by the action stating that the fluent is not changed. However this requires an axiom almost for every pair of fluent and action. Successor state axioms need to be written for every fluent, but they only mention a small number of actions (relevant for this fluent).

5. This question is on planning in general and on goal stack planning.

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

   > There are two new doors installed in the house: the front door and the back door. At the moment they are not painted. The agent can paint a door, provided it is in front of the door. It can also move from one door to the other. In the initial state the agent is by the front door. The goal is to have both doors painted.

   Answer.

   Predicates: $Painted$, At

   Objects: $FrontDoor, BackDoor$

   Initial state: $At(FrontDoor)$ (can also add $FrontDoor \neq BackDoor$, but this usually is inferred from unique name assumption).

   Goal specification: $Painted(FrontDoor) \wedge Painted(BackDoor)$.

   Action schemas:

   $Paint(x)$
   Precondition: $At(x)$
   Effect: $Painted(x)$

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

   (b) Trace the goal stack planning algorithm for this problem. At each step, show what the stack contains, what is the current plan, and the state of the knowledge base if it changes. (15 marks)

   Answer.

   Step 1. The knowledge base is: $\{At(FrontDoor)\}$ The plan is empty.

   Stack is:

   ```
   Painted(FrontDoor) /\ Painted(BackDoor)
   ```

   The top is a compound goal, so we split it and push subgoals on the stack.

   Step 2.

   ```
   Painted(FrontDoor)
   Painted(BackDoor)
   Painted(FrontDoor) /\ Painted(BackDoor)
   ```

   The top is an unsatisfied goal, so we push an action which would achieve it and its preconditions.

   Step 3.

   ```
   At(FrontDoor)
   Paint(FrontDoor)
   Painted(FrontDoor)
   Painted(BackDoor)
   Painted(FrontDoor) /\ Painted(BackDoor)
   ```

The top is satisfied given the $KB = \{At(FrontDoor)\}$. We pop the stack.
Step 4.

```
Paint(FrontDoor)
Painted(FrontDoor)
Painted(BackDoor)
Painted(FrontDoor) /\ Painted(BackDoor)
```

The top of the stack is an action, so we pop the stack, execute the action, add it to the plan, and update the knowledge base with the effect.
Step 5.

```
Painted(FrontDoor)
Painted(BackDoor)
Painted(FrontDoor) /\ Painted(BackDoor)
```

The plan is $Paint(FrontDoor)$.
KB=$\{At(FrontDoor), Painted(FrontDoor)\}$.
The top of the stack is a satisfied goal, so we pop the stack.
Step 6.

```
Painted(BackDoor)
Painted(FrontDoor) /\ Painted(BackDoor)
```

The top is an unsatisfied goal, so we push an action which will achieve it and its preconditions.
Step 6.

```
At(BackDoor)
Paint(BackDoor)
Painted(BackDoor)
Painted(FrontDoor) /\ Painted(BackDoor)
```

The top is an unsatisfied goal, so we push an action which will achieve it and its preconditions.
Step 7.

```
At(x) /\ not (x = Backdoor)
Move(x,Backdoor)
At(BackDoor)
Paint(BackDoor)
Painted(BackDoor)
Painted(FrontDoor) /\ Painted(BackDoor)
```

The top of the stack unifies with the current knowledge base with $x/FrontDoor$, so we pop it.
Step 8.

```
Move(FrontDoor,BackDoor)
At(BackDoor)
Paint(BackDoor)
Painted(BackDoor)
Painted(FrontDoor) /\ Painted(BackDoor)
```

The top of the stack is an action, so we pop the stack, execute the action, add it to the plan, and update the knowledge base with the effect.
Step 9.

```
At(BackDoor)
Paint(BackDoor)
Painted(BackDoor)
Painted(FrontDoor) /\ Painted(BackDoor)
```

The plan is $Paint(FrontDoor), Move(FrontDoor, BackDoor)$.
KB=$\{At(FrontDoor), Painted(FrontDoor), At(BackDoor)\}$.
The top of the stack is a satisfied goal, so we pop the stack.
Step 10.

```
Paint(BackDoor)
Painted(BackDoor)
Painted(FrontDoor) /\ Painted(BackDoor)
```

The top of the stack is an action, so we pop the stack, execute the action, add it to the plan, and update the knowledge base with the effect.
Step 11.

```
Painted(BackDoor)
Painted(FrontDoor) /\ Painted(BackDoor)
```

The plan is $Paint(FrontDoor), Move(FrontDoor, BackDoor), Paint(BackDoor)$.
KB=$\{At(FrontDoor), Painted(FrontDoor), At(BackDoor), Painted(BackDoor)\}$.
The top of the stack is a satisfied goal, so we pop the stack.
Step 12.

```
Painted(FrontDoor) /\ Painted(BackDoor)
```

The top of the stack is a satisfied goal, so we pop the stack.
Step 13.
The stack is empty: stop.
The plan returned is: $Paint(FrontDoor), Move(FrontDoor, BackDoor), Paint(BackDoor)$.

6. This question is on planning in general and on partial order planning.

   (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, (5) goal state (or goal test). A planning problem is formulated in terms of (1) action schemas with preconditions and effects, (2) initial state 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) Suppose there are two objects: $BlockA$ and $BlockB$, and three predicates, $OnTable(x)$, $On(x, y)$, $Clear(x)$. State S_0 is given as follows:

   $$\{OnTable(BlockA), OnTable(BlockB), Clear(BlockA), Clear(BlockB)\}$$

      i. List all the negated ground fluents which are true in S_0. (2 marks)
      Answer. $\neg On(BlockA, BlockB), \neg On(BlockB, BlockA)$.

      ii. Suppose there is an action $Stack(x, y)$ with the following pre- and postconditions:

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

      Give a description of the state resulting from executing $Stack(BlockA, BlockB)$ in state S_0. (3 marks)
      Answer. $\{Clear(BlockA), On(BlockA, BlockB), OnTable(BlockB)\}$.

   (c) Consider the following planning problem. The agent wants to get dressed for cold weather: put on a jacket, a coat and a hat. The predicates are: Hat (meaning, the agent has a hat on), $Jacket$ (meaning, the agent has a jacket on) and $Coat$ (meaning, the agent has a coat on). The agent cannot put on a jacket if it already has a coat on (but can put a coat on top of the jacket).
   The actions are:

   $HatOn$:
   PRECOND:
   EFFECT: Hat

   $JacketOn$:
   PRECOND: $\neg Coat$
   EFFECT: $Jacket$

   $CoatOn$:
   PRECOND:

EFFECT: $Coat$

The initial state is {} (none of the predicates is true). The goal is $Hat \wedge Jacket \wedge Coat$. 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 {}, and the $Finish$ step has $Hat \wedge Jacket \wedge Coat$ as its precondition.
$Links = \{\ \}$
$Orderings = \{Start \prec Finish\}$
Open conditions: $Hat, Jacket, Coat$ (for the $Finish$ step).
Suppose we choose the first open condition as a selected subgoal.
An action which would make it true is $HatOn$. We add it as a step to the partial plan, and a link from it to $Finish$:

$Steps = \{Start, Leave, Finish\}$
$Links = \{\ HatOn \xrightarrow{Hat} Finish\}$
$Orderings = \{Start \prec Finish, Start \prec HatOn, HatOn \prec Finish\}$

Open conditions: $Jacket, Coat$ (for the $Finish$ step).
There are no clobberers.
Suppose we choose the first open condition as a selected subgoal.
An action which would make it true is $JacketOn$. We add it as a step to the partial plan, and a link from it to $Finish$:

$Steps = \{Start, HatOn, JacketOn, Finish\}$
$Links = \{\ HatOn \xrightarrow{Hat} Finish, JacketOn \xrightarrow{Jacket} Finish\}$
$Orderings = \{Start \prec Finish, Start \prec HatOn, HatOn \prec Finish, Start \prec JacketOn, JacketOn \prec Finish\}$
Open conditions: $Coat$ (for the $Finish$ step), $\neg Coat$ (for the $JacketOn$ step).

Suppose we choose the first open condition as a selected subgoal.
An action which would make it true is $CoatOn$. We add it as a step to the partial plan, and a link from it to $Finish$:

$Steps = \{Start, HatOn, JacketOn, CoatOn, Finish\}$
$Links = \{\ HatOn \xrightarrow{Hat} Finish, JacketOn \xrightarrow{Jacket} Finish, CoatOn \xrightarrow{Coat} Finish\}$
$Orderings = \{Start \prec Finish, Start \prec HatOn, HatOn \prec Finish, Start \prec JacketOn, JacketOn \prec Finish, Start \prec CoatOn, CoatOn \prec Finish\}$
Open conditions: $\neg Coat$ (for the $JacketOn$ step).

The remaining open condition $\neg Coat$ is satisfied by the postconditions of $Start$:

$Steps = \{Start, HatOn, JacketOn, CoatOn, Finish\}$
$Links = \{\ HatOn \xrightarrow{Hat} Finish, JacketOn \xrightarrow{Jacket} Finish, CoatOn \xrightarrow{Coat} Finish, Start \xrightarrow{\neg Coat} JacketOn\}$
$Orderings = \{Start \prec Finish, Start \prec HatOn, HatOn \prec Finish, Start \prec JacketOn, JacketOn \prec Finish, Start \prec CoatOn, CoatOn \prec Finish\}$

There is clobbering: $CoatOn$ destroys the precondition of $JacketOn$. So we need to introduce a constraint: $JacketOn \prec CoatOn$:

$Steps = \{Start, HatOn, JacketOn, CoatOn, Finish\}$
$Links = \{ HatOn \xrightarrow{Hat} Finish, JacketOn \xrightarrow{Jacket} Finish, CoatOn \xrightarrow{Coat} Finish, Start \xrightarrow{\neg Coat} JacketOn\}$
$Orderings = \{Start \prec Finish, Start \prec HatOn, HatOn \prec Finish, Start \prec JacketOn, JacketOn \prec Finish, Start \prec CoatOn, CoatOn \prec Finish, JacketOn \prec CoatOn\}$

The plan returned will be $\{Start, HatOn, JacketOn, CoatOn, Finish\}$ with the orderings above.