CS360 Homework 13– Solution

Search

1) Develop an iterative-deepening variant of A* (call Iterative Deepening A*), that is, a version of A* that finds cost-minimal paths for consistent heuristics and uses a series of depth-first searches to keep its memory consumption small. (Hint: If you are stuck, look at the lecture slides on heuristic search.) Solve Problem 3 from Homework 12 for this version of A* with the (consistent) Manhattan distance heuristic.

Answer:
In Iterative Deepening search, each iteration explores the search tree up to a certain depth (given by a depth limit). In Iterative Deepening A* (IDA*) search, each iteration explores the search tree up to a certain f-value (given by an f-value limit). That is, IDA* expands nodes in a depth-first manner, but it does not expand nodes whose f-values are larger than the f-value limit for the current iteration.

Instead of starting with an f-value limit of 0 and increasing it by 1 with every iteration (which might not guarantee optimality if the action costs are not all 1), IDA* first sets the f-value limit to $f(start) = h(start)$ and, for each subsequent iteration, updates the limit to the minimum f-value among the nodes that were generated but not expanded in the previous iteration (that is, the nodes whose f-values are above the f-value limit in the previous iteration but the f-values of whose parents are no larger than the f-value limit). This guarantees that the search finds cost-minimal paths if the heuristics are consistent.

Below is the solution of Problem 1 from Homework 1 using IDA*.

Manhattan distance heuristic:

	A	B	C	D	E
1	2	3	4	5	6
2			3		5
3	g	1	2		4
4	1	2	3		5
5	2	3	4	5	6

Since the f-value of node E1 is 6, we start the first iteration of IDA* with f-value limit 6.

Expansion order in 1st iteration (with f-values): E1 (6), D1 (6), C1 (6), B1 (6), A1 (6, backtrack to C1), C2 (6), C3 (6), B3 (6), A3 (6).

Iterative Deepening A* finds a cost-minimal path in its first iteration.

However, if we were searching from E2 (instead of E1) to A3, it would take two iterations for Iterative Deepening A* to find a cost-minimal path:

Expansion order in 1st iteration (with f-values): E2 (5), (E1 not expanded since $7 > 5$), E3 (5), (E4 not expanded since $7 > 5$).

There are no more nodes to expand, so the 1st iteration terminates without finding a path. Since the smallest f-value among nodes generated but not expanded is 7, we start the 2nd iteration of IDA* with f-value limit 7.

Expansion order in 2nd iteration (with f-values): E2 (5), E1 (7), D1 (7), C1 (7), B1 (7), A1 (7, backtrack to C1), C2 (7), C3 (7), B3 (7), A3 (7).

Search-Based Planning

2) Consider the formulation of the Tower of Hanoi puzzle as a planning problem (solution of Homework 9, problem 2). Using the delete relaxation, calculate the length of the action sequence for achieving each predicate that appears in the goal state from the start state, then calculate the heuristic value of the start state as the maximum of these values.

Answer:

There is only one operator, namely, `Move(X, Y, Z)`. With the delete relaxation, we obtain:

```
Action Move(X,Y,Z):
Preconditions = {Clear(X), On(X,Y), Clear(Z), Smaller(X,Z), Different(Y,Z)};
Effects = {Clear(Y), On(X,Z)};
```

We use the following recursion to compute $h_s(p)$ for all propositions p that appear in the goal state:

h_s(set of predicates s') $= max_{predicate\ p\ in\ s'} h_s(p)$.
h_s(predicate p) $= 0$ if p appears in the state s,
h_s(predicate p) $= 1 + min_{relaxed\ operator\ o\ with\ p\ in\ add\ list} h_s$(precondition list of o), otherwise.

That is, if a proposition p appears in the start state s, $h_s(p) = 0$. The following propositions appear in the start state (for brevity, we skip the propositions of the form `Smaller(X,Y)`, `Different(X,Y)`):

```
Clear(D1), On(D1, D2), On(D2, D3), On(D3, P1),
Clear(P2), Clear(P3).
```

Let P_i denote the set of propositions p with $h_s(p) = i$. That is, P_0 contains exactly the propositions in the start state. Note that P_0 contains `On(D1, D2)` and `On(D2, D3)`, but not `On(D3, D3)`. We now incrementally compute P_1, P_2, ..., until we discover the P_i that contains `On(D3, D3)`.

We start with P_1. This is the set of propositions that can be added by an operator whose preconditions are satisfied by P_0. There are only two operators whose preconditions are satisfied by the propositions in P_0, namely, `Move(D1, D2, P2)` and `Move(D1, D2, P3)`. The add effects of these two operators have the propositions `On(D1, P2)`, `On(D1, P3)`, `Clear(D2)`. Since none of them are in P_0 and all preconditions of the operators that add them are in P_0, all of them must be in P_1 (because of the recursive rules listed above). Therefore, $P_1 = \{\texttt{On(D1,P2)}, \texttt{On(D1,P3)}, \texttt{Clear(D2)}\}$. Below, we list $P_0 \cup P_1$.

```
Clear(D1), On(D1, D2), On(D1, P2), On(D1, P3),
Clear(D2), On(D2, D3), On(D3, P1),
Clear(P2), Clear(P3).
```

We now identify P_2. This is the set of propositions that can be added by an operator whose preconditions are satisfied by $P_0 \cup P_1$. There are two new operators whose preconditions are satisfied by $P_0 \cup P_1$, namely `Move(D2, D3, P2)` and `Move(D2, D3, P3)`, which add the propositions `On(D2, P2)`, `On(D2, P3)`, `Clear(D3)` (note that, at this point, any operator that moves D1 does not add any new propositions, so we skip those operators). These propositions do not appear in $P_0 \cup P_1$ but can be added by operators whose preconditions are in $P_0 \cup P_1$. Therefore, $P_2 = \{\texttt{On(D2,P2)}, \texttt{On(D2,P3)}, \texttt{Clear(D3)}\}$. Below, we list $P_0 \cup P_1 \cup P_2$.

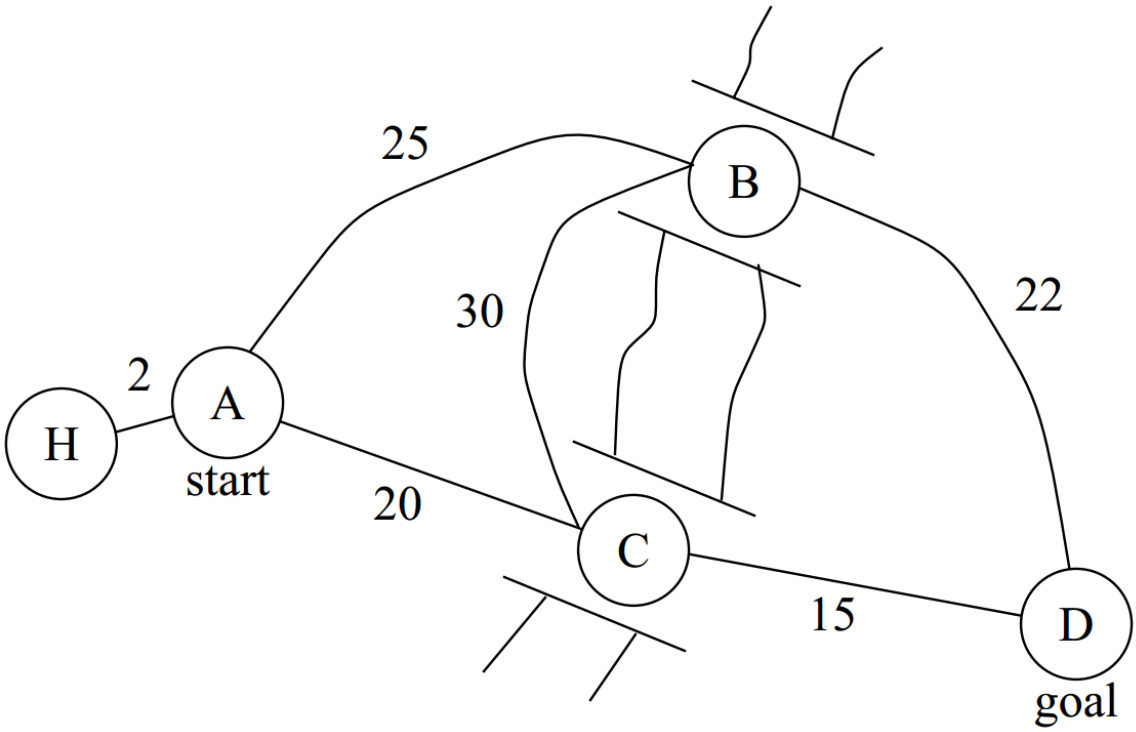
```
Clear(D1), On(D1, D2), On(D1, P2), On(D1, P3),
Clear(D2), On(D2, D3), On(D2, P2), On(D2, P3),
Clear(D3), On(D3, P1),
Clear(P2), Clear(P3).
```

$P_0 \cup P_1 \cup P_2$ satisfy the preconditions of `Move(D3, P1, P3)`, which adds `On(D3, P3)`. Therefore, `On(D3, P3)` $\in P_3$ and, consequently $h_s(\texttt{On(D3,P3)} = 3$.

Taking the maximum of 0, 0 and 3, we obtain 3 as the heuristic value of the start state.

Value of Information

3) A mobile robot is trying to get from his current position A to a destination D as quickly as possible. There is a river separating A from D and there are two bridges, B and C, spanning the river. The robot must design a strategy to move from A to D via one of the bridges.



Though the robot knows that one (and only one) of the bridges is inoperable, it is uncertain regarding which one of the two bridges is out. From its start position, position A, the robot can climb a hill to position H and use sensors to obtain information regarding which bridge is out. Collecting this information will take time, since it must go to H and return back to A, and the information gained is uncertain because the sensors will not be able to tell precisely which bridge is out.

The numbers in the above graph give the distance between locations in miles. In similar situations in the past, the robot experienced that 4 out of 5 times bridge C was out and only 1 out of 5 times bridge B was out. The robot has a short-range sensor that tells it with 100 percent reliability whether a bridge is out. The sensor can only be used when the robot is directly in front of the bridge. The long-range sensor of the robot is unreliable. It errs with a probability of 10 percent, that is, suggests that the broken bridge is operable and the other bridge is broken.

Design a strategy for the robot that minimizes the expected execution time.

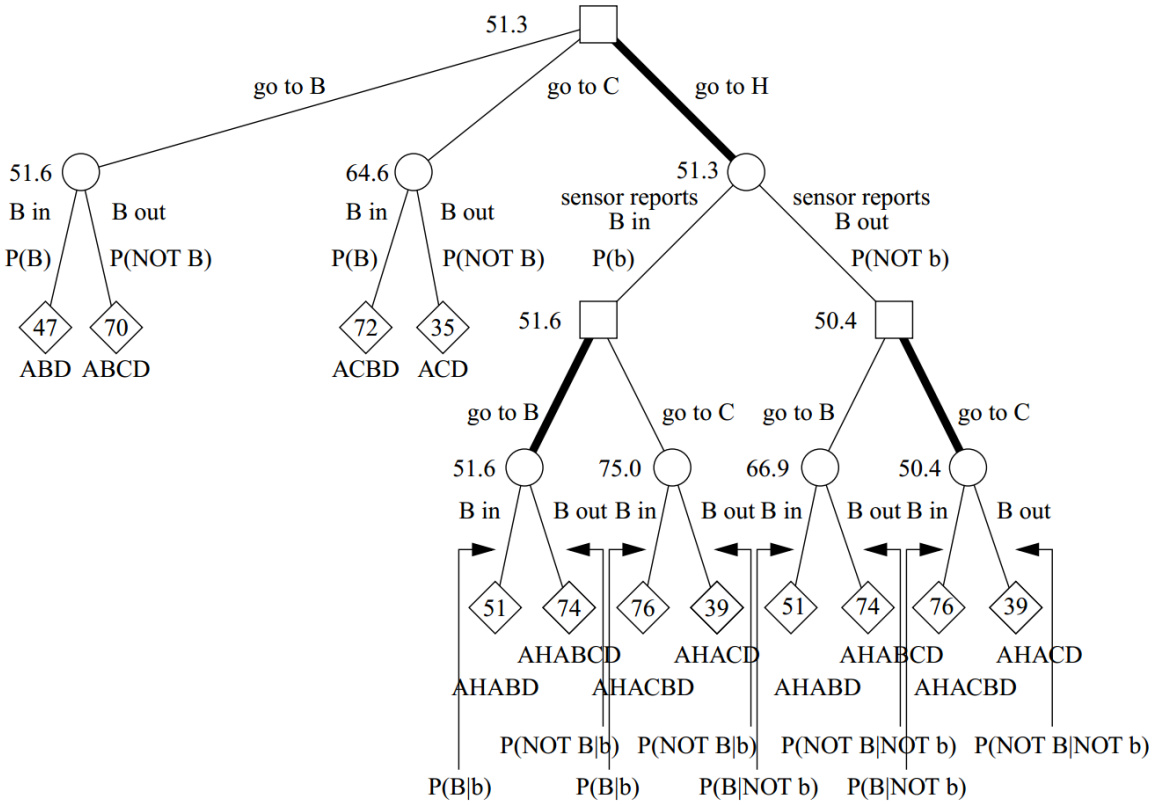
Answer:
The plan that minimizes the expected execution time is the following: the robot should go from A to H and use its long-range sensor. If the long-range sensor reports that bridge B is operational, the robot should then go to B via A and, if the bridge is indeed operational, continue to D. If the bridge is not operational, the robot has to continue to D via C. If the long-range sensor reports that bridge B is not operational, the robot should go to C via A and, if the bridge is indeed operational, continue to D. If the bridge is not operational, the robot has to

continue to D via B. The expected execution time of this conditional plan is 51.3 minutes. The value of information of climbing the hill is 51.6 - 51.3 + 4.0 = 4.3 minutes.

The calculations are as follows:

```
B = bridge B is operable (and bridge C is broken)
C = bridge C is operable (and bridge B is broken)
b = long range sensor reports that bridge B is operable (and bridge
C is broken)
c = long range sensor reports that bridge C is operable (and bridge
B is broken)

P(B) = P(NOT C) = 4/5 = 0.8000
P(NOT B) = P(C) = 1/5 = 0.2000
P(NOT b | B) = P(c | B) = 0.1000
P(b | B) = P(NOT c | B) = P(b | NOT C) = P(NOT c | NOT C) = 0.9000
P(b | NOT B) = P(NOT c | NOT B) = P(b | C) = P(NOT c | C) = 0.1000
P(NOT b | NOT B) = P(c | NOT B) = P(NOT b | C) = P(c | C) = 0.9000
P(NOT b | B) = P(c | B) = P(NOT b | NOT C) = P(c | NOT C) = 0.1000
P(b) = P(b AND B) + P(b AND NOT B)
     = P(b | B) P(B) + P(b | NOT B) P(NOT B) = 0.7400
P(NOT b) = 1 - P(b) = 0.2600
P(B | b) = P(b | B) P(B) / P(b) = 0.9730
P(NOT B | b) = 1 - P(B | b) = 0.0270
P(B | NOT b) = P(NOT b | B) P(B) / P(NOT b) = 0.3077
P(NOT B | NOT b) = 1 - P(B | NOT b) = 0.6923
```

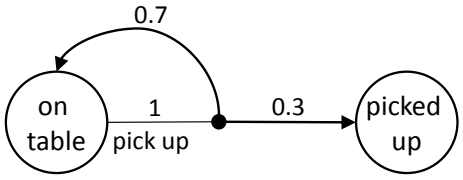


Markov Decision Processes

4) Assume that you are trying to pick up a block from the table. You drop it accidentally with probability 0.7 while trying to pick it up. If this happens, you try again to pick it up. How many attempts does it take on average before you pick up the block successfully?

Answer:

We can model this problem as an MDP as follows:



Now, we can calculate the expected cost with $X = 1 + 0.7X + 0.3Y$ and $Y = 0$ (where X = on table and Y = picked up), which gives us $X = 1/0.3 \sim 3.33$.

The Markov assumption is the assumption that, whenever one executes an action in a state, the probability distribution over its outcomes is the same, no matter how one got into the state. Is this assumption realistic here?

No. If the pickup operation does not work the first time (e.g. because the block is too slippery), it will likely not work the next time either.