Massachusetts Institute of Technology

6.034 Artificial Intelligence

Solutions # **Quiz1** 6034 Item # **16**

Class average was 73, std. dev. was 11.8

Problem 1 Rules (20 points)

Part A (8 points)

There are three triggered rule instances:
Rule 1: $?x \mapsto B1, ?y \mapsto B1$
Rule 1: $?x \mapsto B2, ?y \mapsto B2$
Rule 1: $?x \mapsto B2, ?y \mapsto B1$

Note that Rule 2 is automatically precluded on acount of the second AND-IF condition, since everything is a block.

Part B (12 points)

1. Answers: (a), (b), (c).
   Comment: The important part was to get (a) and (b) right. The reason why (c) also matters is slightly more subtle; changing the order of the patterns might affect how variables are instantiated, which might affect which rules become triggered.

2. Answer: (e).

3. Answers: (a), (b), (c), (d).
   Comment: Here the important part was to get (a) and (d) right.

4. Answers: (a), (b), (c), and (d).
   All general-purpose rule-based systems are susceptible to infinite loops, e.g. by having rules whose IF part matches their own THEN part.

Problem 2 Search (20 points)

We can convert this city map into a search tree. Note that taking branch 2 leads you to the goal G after passing through 13 intermediate nodes. However, taking branch 1 brings you to a subtree of branching factor b=2 and depth d=9 starting at the leftmost node of the lattice-like network (we call this leftmost node "node A") and ending at the rightmost node of the lattice-like network (let's call this node "node B"). Hence, node A corresponds to the root node of this subtree and node B corresponds to all of the leaf nodes of the subtree.

Notice that the total number of possible paths passing through place I equals the number of leaf nodes of this subtree. This is because place I follows directly after node B. Further notice that all paths passing through place I lead back to node A, thereby forming a cyclic path. Paths with cycles are not allowed, so whenever a cyclic path is formed the search procedure fails and must backtrack to find a different path to G.

Recall: In a tree of branching factor b and depth d, there are b^d leaf nodes.

Part A (4 points)

Answer: 2^9

In depth first search, there is no heuristic determining which branch to take. Hence, we are instructed to take branch 1. As a result, we encounter the subtree beginning at node A. Every path of the subtree passes through place I, but eventually bites its own tail when it hits node A again. Hence, before we can backtrack to branch 2, we must explore (and fail at) every path of this subtree. There are 2^9 paths total, so we encounter 2^9 paths pasing through place I before reaching the goal.

Part B (4 points)

Answer: 2^9

Notice that node A is closer to goal G than the node following branch 2. As a result, the hill climbing procedure proceeds in a depth first manner down branch 1, completely exploring the subtree before taking branch 2 to the goal. As in part a, place I is encountered 2^9 times.

Part C (4 points)

Answer: 2^9

Best-first search will explore paths in order of the estimated distance of their end node from the goal. All the paths down branch 1, as well as the one down branch 2, are explored. But, all the paths through I move closer to the goal (by straight-line distance) than the path to the goal down branch 2 so they get looked at first before finally finding the complete path to the goal.

Part D (4 points)

Answer: 2^9

In breadth first search, we will explore all paths with i nodes before looking at paths with $i + i$ nodes. So, we will explore all the paths down branch 1 before reaching the goal down branch 2 (since the goal is deeper in the search tree than I is).

Part E (4 points)

Answer: 1

No heuristic is involved in determining which branch to take if we augment depth first search with the dynamic programming principle. Therefore, we proceed down branch 1 before branch 2. Because the dynamic programming principle prunes all paths that are longer than or equal to in length of the current path reaching the same node, only one path makes it though node B. This path passes through place I before it hits node A , forming a cycle and causing search to fail. Then, branch 2 is taken to the goal.

Problem 3 Search (20 points)

Part A (4 points)

What measure corresponds to "distance travelled" for this problem?

Answer: The layout length L is what we are trying to minimize, this is the analog of "distance travelled."

Part B (8 points)

Let us redefine hill-climbing as an attempt to find quickly a good, but not necessarily optimal, path.

Circle all the measures below that would help hill-climbing search find reasonably good layouts. Write "best" next to the measure that would be likely to find the best solution. Assume each search node represents a partial layout with $k \le n$ pieces.

Answer: Hill climbing picks the node (among descendants of current node) that minimizes the value of some measure, typically distance to goal, but, in our case, some measure of quality of layout. The correct answers are (1) and (2) with (1) being best.

1. This is precisely what we are minimizing, so this is the best measure to guide us. Hill climbing will be "greedy" and not guarantee finding the best answer, as branch and bound or A^* would.
$$L = \max_{i \le k}(x_i + L_i)$$
2. Almost L, this is the next best.
$$\max_{i \le k} x_i$$
3. Minimizing this measure leads to terrible layouts with all the parts along one border.
$$\max_{i \le k} y_i$$
4. This is subtler, this measure has minimum value 0. This value is reached by making U big, bigger than R. These are, in general, terrible layouts.
$$\max(0, R - U)$$
5. Same as above.
$$\frac{\max(0, R - U)}{W}$$

Part C (8 points)

Circle the measures below that can serve as valid underestimates of distance to the goal (from a given partial layout with k pieces) for A^*. Write "best" next to the measure that be likely to find the optimal solution fastest. Assume each search node represents a partial layout with $k \le n$ pieces.

A^* uses "distance travelled," i.e. L PLUS an understimate to minimize. So, using L makes sure we get the best answer. The understimate speeds our search. The correct answers are (1) and (4) with (4) best.

1. The trivial underestimate. Can always be used with A^* and we get basically branch and bound with dynamic programming.
$$0$$

2. This is not an understimate, often you may add a piece without increasing the total length of the layout.

$$\min_{k<i\leq n} L_i$$

3. This is a measure of area, not length. When added to L, it may make the optimal path look worse than it is.

$$\max(0, R - U)$$

4. This is the smallest possible value of additional length required. Think of R as a liquid being poured into a container of area U, $R - U$ is the area left over (if any), dividing by W gives the length.

$$\frac{\max(0, R - U)}{W}$$

Problem 4 Constraint Propagation (20 points)

Part A (5 points)

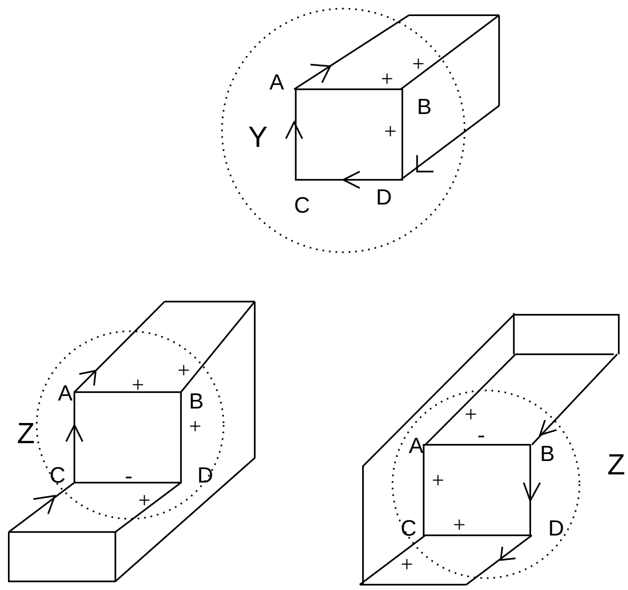

Figure 1: Sample Solutions

Part B (15 points)

Suppose you have tree search algorithm that finds all ways to label junctions such that each junction has one junction label and that junction label is consistent with neighboring junction labels.

For X, the tree search algorithm would find n legal combinations of junction labels where n is (check off the correct answer):

X	zero
	1
	more than 1

For Y, the tree search algorithm would find n legal combinations of junction labels where n is (check off the correct answer):

	zero
X	1
	more than 1

For Z, the tree search algorithm would find n legal combinations of junction labels where n is (check off the correct answer):

	zero
	1
	the product of the number of labels at each junction
	the sum of the number of labels at each junction
X	none of the above

May be any number between 0 and the product of all the remaining junction labels

Problem 5 Search with Constraint Propagation (20 points)

Part A (6 points)

Ignoring the constraint that intersecting words must have the same letter lat the place of intersection, there are 8^{13} **(8 (8 letter) words for 2 slots by 8 (4 letter) words for 11 slots)** combinations of words that would fit into the crossword puzzle.

Part B (6 points)

After some thought, you conclude that this kind of crossword puzzle seems much like the map coloring, line labelling, and schedule planning problems that appeared in Problem Set 4. In particular, in the crossword puzzle, the variables correspond to

The slots in the puzzle that the words go into.

the variable domains correspond to

The words of the correct length to fit into the appropriate slot. (Note: you could also just say "the words" here, and then specify word and slot length as another constraint below.)

and the constraints correspond to

"Intersecting words must have the same letter at the place of intersection," (to quote Part A.)

Part C (8 points)

Now, you decide to solve the crossword puzzle using backtracking search, inserting words in places in the following order, making sure that each word is both of the right length and matches all words previously inserted at points of intersection:

1 down—2 across—3 down—3 across—4 down—5 across—6 down—6 across—7 down—8 across—9 down—9 across—10 down

The question is, does constraint propagation (such as embodied in AC-3 or AC-4 in Problem Set 4)—which was used in solving the map coloring, line labelling, and schedule planning problems—help or not help in doing the crossword puzzle?

Answer 1: In certain map coloring problems, constraint propagation cuts down on search enormously. If you think constraint propagation *cannot* cut down on the search enormously in working crossword puzzles of the sort described in this problem, carefully explain the *one key difference* between maps and this kind of crossword puzzle:

Anything here was wrong, but we gave it one point because it was better than no answer at all!

Answer 2: If you think constraint propagation *will* cut down on search enormously, fill in a sequence in the chart below—starting with the choice at 1 down and continuing as far as necessary—such that backtracking search would continue, but search pruned with constraint propagation would backtrack.

1 down	**flummery**
2 across	
3 down	
3 across	
4 down	
5 across	
6 down	
6 across	
7 down	
8 across	
9 down	
9 across	
10 down	

The point here was to show the point where constraint prop would stop, but backtracking would keep going. Showing how far backtracking would go without showing where constraint prop would stop cost you a point. We were *not* looking for you to fill in the right answer! (BTW, it started with "abbatoir," for those of you who didn't take time to work it out.)