Midterm

CSC 242

2 March 2000

*Write your **NAME** legibly on the bluebook. Work all problems. Best strategy is not to spend more than the indicated time on any question (minutes = points). Open book, open notes.*

1. Reactive Agents: 10 Mins. You are designing a vacuum-cleaning robot to sweep your house while you are away. You decide to make it a reflex agent with internal state. What sort of rules or reflexes should it have and what sort of sensors do they presuppose? Suppose you want it to be able to go dump its dust load when full, or to go to a recharging station when its batteries are low. Is the reflex agent an adequate architecture and if not what must be added?

Answer: Might have a wandering rule to have it move randomly around, or could just have it go forward until it gets near to or actually hits something. A reflex to have machine back up and reorient itself if hits or nears obstacle. Could have rule to keep it on certain sorts of floors, so say don't cross lino-rug boundary or vice-versa. One decision would be whether to have the vacuum on all the time or to try to detect dust: the first option is expensive in power, the second is a hard practical problem. You might try to put in rules to have the robot do a systematic job of covering the floor, or you might rely on chance, as in a swimming-pool-cleaning device. Realistic sensors: infra-red proximity sensors, touch sensor, possibly some downward-looking reflectivity sensor to tell floor type. "Dust sensor" is tough...

Anything involving going to some particular spot, like a dustbin or a recharging station, involves more state (the map of where things are and how to get there) than a simple reflex agent can contain. That is the next level up, in which goals are added. At that point the robot has its choice of things to do (dust, dump, recharge...) and knows how to accomplish each of several things.

2. Search Combinatorics: 10 Mins. Fig. 3.18 of the text (attached) asserts that a depth-limited search requires time b^l, space bl, is not optimal but is complete if $l \geq d$. (b is the branching factor, d the depth of solution, and l the depth limit). Please justify all these assertions.

Answer: Depth-limited is like depth-first, so it in the worst case it explores all the solutions down to a given depth (in depth-first order), but does so only remembering the current path from the root it is depth-firsting on. That means that in the worst case it has to look at b^l nodes, namely all the nodes in a tree of depth l and branching factor b, so the time it spends is proportional to that. If it were breadth-first it would also have to remember all those nodes, so its space requirements would also be proportional to b^l, but it's not: it only needs to remember the nodes explored or not at each level down to level l, and there are only bl such nodes to remember. The depth-first type search may find a solution first that is not optimal, so depth-limited shares non-optimality with depth-first search. If the depth of a solution lies within the depth limit, then it will be found, since the algorithm will be forced to look down all paths to that depth. Pure dumb depth first

search can just dive down the first bad path it sees and miss good solutions at shallow depths. Thus depth-limited is complete if the depth of the solution is less than its depth limit.

3. Search Strategies: 15 Mins. First (5 mins), give an example of a problem in which depth-first search would work better than breadth-first and say why. Second (5 mins), give an example of a problem in which breadth-first search would work better than depth-first and say why. Third (5 mins), when would best-first search be better than breadth-first?

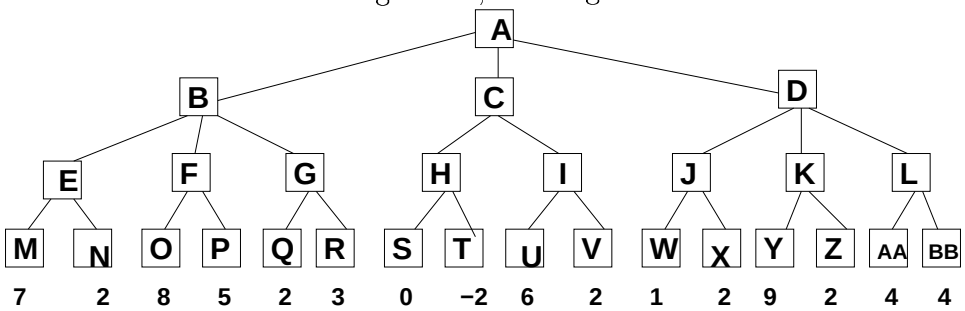
Answer 1: Depth beats Breadth: if limited memory requirements or big branching factors, like in Chess end games where deep lookaheads are needed, or planning a trip from A to B.

Answer 2: Breadth beats Depth: if memory not a problem and it's not easy to detect cycles or dead ends, as in finding your way out of a maze. Often puzzle-scale problems are amenable to BFS.

Answer 3: Best-first beats breadth in any number of problems solvable by greedy methods, like minimum spanning tree, shortest-path. Or for practical but non-optimal solutions to NP-hard problems like TSP.

4. Minimax and $\alpha - \beta$: 10 Mins. Consider the following game tree in which the indicated static scores are all from the first player (MAX)'s point of view. What move should MAX choose (3 mins)?

Answer: MAX should go to B, which guarantees MAX a 3.



Which nodes would not be examined if alpha-beta pruning is used (7 mins)?

Answer: P, T, I (and its children U and V), K (and kids Y,Z), L (and kids AA, BB).

6. Resolution Proof: 10 Min. Given these premises:
1. Anyone who is not addled is balmy.
2. Not everyone is both balmy and crabby.
3. Everyone is crabby.

Express them in FOPC, put them into clause form, and use resolution to answer the question: Who is addled? (or: Is anybody addled, and if so who is he?).

Answer: The premises go to: (after implication elimination in the first)

$$(B(x) \vee A(x))$$

$$\exists x(\neg B(x) \vee \neg C(x)) \longrightarrow (\neg B(a) \vee \neg C(a))$$

$$C(y)$$

and we want to prove that there exists an addled person. So we assert that and then deny it and clause-ify it, and seek a contradiction (from $\neg A(x)$, which we label clause 4.) Relabelling variables, we can resolve 3 with 2, then 2 with 1, carrying along the skolem constant to put 2 into clause form, and then resolve 4 with the result of that resolution, which derives the null clause and proves that there is some addled person, and in fact it is the one that is not both balmy and crabby.

7. Heuristic Search: 10 Min. The traveling salesman problem is to visit every city in a set of cities exactly once, returning to the start, covering the shortest distance on the ground. We'd like to set up the search for the solution route as an A* search over the cities. That is, we are going to do best-first search with an evaluation function

$$f(n) = g(n) + h(n),$$

where $g(n)$ gives the cost of the path so far, which will just be the distance the salesman has driven. Which, if any, of these choices for $h(n)$ guarantee that the shortest path is found (are admissible heuristics?) Assume that at city n there are M cities left to visit.

1. $h(n) = M$ times the maximum distance between any two unvisited cities (3 mins).
2. $h(n) = M$ times the average distance between all pairs of unvisited cities (4 mins).
3. $h(n) = M$ times the minimum distance between any two unvisited cities (3 mins).

Answer: 1. is not admissible, it overestimates. 3. is admissible, it never overestimates 2. is not admissible: it can overestimate (e.g. if each city has a long and a short route between each other one) or underestimate (e.g. if there are a few long necessary routes and lots of short redundant ones).

8. Hierarchical Planning: 10 Mins. First (3 mins), explain the role of the upward and downward solution properties in hierarchical planning. Second (4 mins), In the example attached (Fig. 12.5), we see "We assume that $1/b$ decompositions will lead to a solution." Is this a reasonable assumption and why or why not? Third (3 mins) If $2/b$ decompositions lead to a solution, is that better or worse for hierarchical planning and why?

Answer: If these properties can be guaranteed it means that abstractions may be substituted for fully-expanded plans when it comes to predicting whether a plan exists that will achieve what the abstract plan does, and whether a flawed abstract plan guarantees no workable detailed plan. Without these properties, you cannot prune your plan search space confidently, and thus you lose the advantages of hierarchical decomposition.

The $1/b$ assumption is that all but one of the abstract plans created to achieve a step will be inconsistent and can be discarded. This is a "best case" for hierarchical planning, and seems a little unlikely given that there are possibly redundant or overlapping operators, more than one way for effects to be achieved, etc.

If the downward solution property holds, more decompositions leading to a solution are no worse: since you know any one will work, prune all the others. If you can't guarantee downward solution property, then $2/b$ doubles your branching factor and is much worse.