Answer Key – 1st Midterm Exam

Tuesday February 19, 2008
75 minutes == 75 points
Open book and notes

1. 20 points
You are given an instance of the traveling salesperson problem (TSP). A salesperson has to visit a group of cities, visiting each only once and getting back to the starting city. The objective is to minimize the total distance traveled. Assume each city is directly connected to each other city.

1.1. Describe a state-space representation for the problem, specifying initial state, goal state(s), and operators.

Correct Answer
There was some confusion as to what a state it. A state represents the state of the problem at a specific point in time. For example, on a chess board, a state might be 64 tiles with a queen in the upper left corner and a king in the bottom right. It is not the algorithm used to generate that state. For this question, it was important to understand the difference between a state and the other aspects of a problem (for example, the algorithm that selects/generates states). The two most common ways to model a state is complete-state and incremental (for a review, see the n-queens discussion on page 66 in the book).

In the incremental representation, a state is the set of cities that have been visited. The initial state is the starting city. The goal state is that all cities have been visited. The "go to city" operator adds a city to the state.

The alternative is a complete-state representation, where the state always contains all cities. For example, for n cities, the state is [1,2,…,n] where each spot represents a city. The array represents the order cities are visited and thus it is both a state and a solution. The initial state has the first element as the starting city and the other elements the other cities, chosen in any order. The goal state is that all cities have been visited (and the salesperson has returned to the start city) on the shortest path. The operators swap two cities.

Since in both representations the state represents a path, the first element must be the starting city. The goal state is the solution, so there is no need to return the solution path.

Due to the confusion many students had on successor functions, the exam asked only for operators. The only necessary operator was to move the sales person from one city to another. The constraint on the operator is that it can only move to city that has not yet been visited, with the exception of the starting city, and then only when it is the last city.

Scoring
For the state representation, the state had to be specific enough that a programmer would know how to implement it and you would know how to calculate the number of possible states for question 1.2. It did not need to be in mathematical or code format but it had to be unambiguous. Since the goal test will check the current state against the goal state to see if a solution has been found, the state representation must be in a form where it can be compared to the goal test.

Question 1.1 was worth 5 points.

1 point was taken off if the state representation contained inappropriate (non-state) information, duplicated information that was already stored on the map (e.g., connectivity or distances), was vague or could not be tested against the goal condition.

1 point was taken off if the initial state did not contain the start node. Without this, a search algorithm cannot come up with a solution.

1 point was taken off if the goal state did not contain all the cities that had been visited.

1 point was taken off if the state operators did not contain a movement operator, included algorithm (rather than state) operators (e.g.,

backtracking) or was ambiguous (e.g., look at one city, move to another).

2 points were taken off if any answers was completely wrong (e.g., the initial state is a map, salesman and a set of actions; operator is to find the shortest path; goal state is to visit all cities once). No more than four points could be lost on this question if a good faith effort was made.

Common Answers

Initial:

1. Map of all cities. Starting city flagged as containing the sales person.
2. Traveling sales person is in the initial city. No cities have been visited.
3. With n cities, represent state as an (n-1)-tuple ($b_1,...b_{n-1}$) of Booleans representing whether the city has been visited. The last city is n, the same as the start city. Start state is (0, 0,...0).
4. Picture of an array. First two elements are current city and starting city. Each remaining element is whether the city has been visited. Initial state is (S, S, N,... Y,...N) where the Y is in the slot for the starting city.
5. All cities unvisited, 0 distance traveled.
6. And adjacency matrix, each city represented by its index/name and a Boolean variable "visited". All cities visited=false, currentCity=S.
7. {start city}. A list of cities already visited.
8. [starting city, [nil], next city, nil]. For [city, [cities visited list], best city to move to, previous].
9. Any city.
10. In(some_city).
11. start city, distance 0. For state rep locations, distance.
12. (k, nil). For state rep (city_name, visited) where visited is a set of cities visited.
13. (home city). A tree representation of each city, the cost between each city and their connected cities.
14. No cities visited.
15. In(<starting city>), Visited(<starting city>)
16. Ordered n-tuple, k that is some obvious solution and a real number d(k) which would be the total distance traveled if k elements would be visited chronologically.
17. A group of cities labeled uniquely on a map, a traveling salesman with a pre-determined mode of travel at the starting city.

Goal:

1. All cities flagged as visited. Sales person in starting city.
2. (1, 1,...1) and then visiting the start city.
3. Picture of an array. (S, S, Y,... Y) where the Y is in the slot for.
4. All cities visited, shortest distance traveled. (*i don't know if we want to encourage people to put performance measurements in their goal state*).
5. All cities visited=true, currentCity=S.
6. {start city, city 1, ... city n}
7. [starting city, [list of all cities], nil, last city].
8. (k, N). For state rep (city_name, visited) where visited is a set of cities visited.
9. k:d(k)<e for some e>0.
10. The shortest tour visiting all cities exactly once except the starting city.

Operators:

1. Remove sales person from city. Mark city as visited. Mark city containing sales person (no action to move sales person).
2. Go to next unvisited city (change a 0 to 1). From start state, n-1 operations are possible, then n-2, etc.
3. Visit city i given that that city i has not been visited yet (other than return home).
4. Find unvisited cities, find shortest path to a given city, travel to city.
5. GoToCity(i), only if i.visited=F. i=S is terminal condition (search ends).
6. go-from<city x, city y>
7. Look at distance to next city, move to another city.
8. Choose shortest path, cannot visit a city twice, start city is the goal city.
9. Move to a new city or backtrack.
10. Go(<destination city>), In(<current city>).
11. Swap the positions of two elements of k.
12. Salesman moves from one city to the next using shortest Euclidean distance without visiting the same city until returning to the starting

city

1.2. For a problem with n cities, how many states are there in the search space? Explain briefly your answer.

Correct Answer
The number of states depends on how you represented your state, normally either a complete-state formulation (state has all cities in it) or an incremental state formulation (start with one city, add a city each action).

For a complete-state formulation the number of states is (n-1)!
For a five city TSP where you must start in city A, there are the states using complete-state.

ABCDE	ACBDE	ADBCE	AEBCD
ABCED	ACBED	ADBEC	AEBDC
ABDCE	ACDBE	ADCBE	AECBD
ABDEC	ACDEB	ADCEB	AECDB
ABECD	ACEBD	ADECB	AEDBC
ABEDC	ACEDB	ADEBC	AEDCB

I find this representation confusing, but is enough students used it it is ok to show it. In this representation, how do they keep track of the path followed? If your representation did not track the order the cities were visited (e.g., an array of Booleans representing whether a city had been visited), there are states.

FFFF	FFFT	FFTF	FFTT
FTFF	FTFT	FTTF	FTTT
TFFF	TFFT	TFTF	TFTT
TTFF	TTFT	TTTF	TTTT

For the incremental representation, the number of state is larger since states represent partial paths, but the number of solutions (i.e. of complete paths) is the same, (n-1)!
For a five city TSP where you must start in city A, there are 24 = (n-1)! paths (i.e. leaf nodes on the tree). (note that we are not including the "move back to start city" here, which would not add any additional solution).

Scoring
Very few students got this question correct. Those who got it wrong got it wrong differently – in the first 27 exams i counted 14 different answers.

For a five city problem, answers ranged from n-1 (number of states=4) to $n!^n$ (number of states= 24,883,200,000).

0 points were taken off if you used an incremental formulation but described the number of paths rather than the number of states.

1 point was taken off for forgetting to account for the initial city (using n! rather than (n-1)!).

3 points was taken off for using a formula that was wrong but had a decent explanation.

4 points was taken off for using a formula that was obviously wrong (e.g., n).

Common Answers

1. $(n!)^n$. n! is one path. ^{n} represents that circuit through any given point.
2. 2^{n-1} ignoring going back to the start state. Each state in b_i in (b_1,...b_{n-1}) are Booleans and there are no loops.
3. since each city is directly connection to each other (full graph). Number of states is the same as the number of edges.
4. $n2^n$ since each city has been visited or not and there are n cities. The first n is if the starting city is arbitrary.
5. n-1. The sales person begins at one of the n cities and therefore doesn't have to travel to that one in the beginning.
6. (n-1)!. From the start city, you could go to (n-1) other cities from there to (n-2) cities, etc.=(n-1)!
7. n!
8. n*branching factor.
9. n^2. You can be in any of n cities and have visited up to n cities before it.
10. $O(n^2)$. Because the starting city is undefined so all the nodes must be considered starting.
11. 2^n. But it might be possible that not all states are considered or taken into account.
12. n+n!. Represents number of cities visited (n!) and the cities for the trip back (n).
13. n. You would start In(1)->In(2)->...In(n).
14. m^d. Same as the minimal spanning tree problem.

1.3. Suppose you use A* with the following heuristics: in each city h(n) is the distance to the closest city not yet visited. Is this heuristics admissible? Explain briefly your answer.

Correct Answer

Yes, to be admissible you must not overestimate the remaining cost to the goal state and since you must visit every node in a TSP (touring) problem, the distance to the closest unvisited city can never be more than the distance to that and all other unvisited nodes.

Scoring

0 points were taken off if you failed to mention that, since this is a touring problem you were required to visit the closest node.

1 point was taken off for giving the right answer plus additional information that was incorrect.

2 points were taken off if you said the heuristic underestimates but did not explain why.

4 points were taken off if you said the heuristic was not admissible.

Common Answers

1. Yes. Since h(n) is the smallest distance you can travel to a city from your point, it is admissible because you cannot overestimate the distance.
2. Yes. Total cost to goal is at least the cost to the next node so cannot overestimate.
3. Yes. It underestimates. (*no explanation as to why he believes this*)
4. Yes. At any given state, the TSP must travel to an unvisited city and then back to the start city, which is obviously > h(n). If there are no unvisited cities, h(n)=0 which is an underestimate.
5. Yes. All it does is search through all possible unvisited cities, finds the shorted path to each and returns the shortest path. (*i don't even know how to begin scoring this answer*)
6. Yes. It is the best case solution at that point in time. If the distance is the shortest than we have to do at least that much work to get to the next city.

7. It fails to take into account the distance from the last city to the starting city. The admissibility depends on how strictly you follow the definition.
8. No. Because h(n) will never overestimate since every city must be visited, however, there could be a shorter path to completion.

1.4. Suppose you use the above heuristics in a greedy best-first search algorithm. Are you guaranteed to find an optimal solution? Explain briefly your answer.

Correct Answer
No, greedy best first search only looks at the cost to the next node (the local cost), does not include distance traveled (and thus can't calculate total path costs) and does not backtrack if the total path cost exceeds the potential cost of an alternative path.

Drawing a picture always helps prove one's case (diagram taken from an excellent student response):

Starting at A, greedy best first goes to B (cost=1) then C (cost=1), forcing it to go next to D (cost=1) and then back to A (cost=100). The total cost is 103. A better path is ACDBA with a cost of 6.

Scoring
0 points was taken off for stating that it would not find the optimal answer because it would get trapped in a loop. This was not the desired answer. Whether BFS gets into loops depends on the implementation – when used with a closed list, this is not an issue. But since the problem used the word "guaranteed" and some implementations are susceptible to cycles, I decided to give credit for it as long as a detailed explanation was given. But understand that this is not the core problem.

1 point was taken off for giving the right answer plus additional information that was incorrect.

3 points were taken off if you asserted that greedy best first was not complete but did not explain why, or your explanation was vague.

4 points were taken off if you said BFS would return an optimal path, gave an explanation that was significantly incorrect or showed a significant lack of understanding of solving problems with search algorithms (e.g., greedy search only looks at one move at a time but to be optimal you should plan your entire trip at once).

Common Answers
1. No. In the diagram above, Greedy Best First would choose A-B-C-D=103. The optimal path is A-C-D-B=6. (*several people drew diagrams, which i appreciate*).
2. No. Greedy chooses the immediate best answer, ignoring the cost up to now (g). As such, something with a high g(n) might include that path if it has children with a small cost.
3. No. Choosing a city early on can lead to the search missing shorter total paths and can potentially lock them out of visiting cities that could use those path later.
4. No. Would return to cities it's already been too which would not be optimal. Greedy searches are only concerned with the best choice for one move at a time and it might not be the best way and to be optimal the entire trip should be planned at once.
5. No. It neglects the distance covered so far and only considers the heuristic function.
6. No. There might be some path in a larger node that never got expanded that contains a more optimal solution. It is similar to depth first and is not optimal and incomplete. It can also get stuck.
7. No. It ignores the cost of getting home and thus may find a poor solution compared to A*.
8. No. After it visits all cities, h(n) will go to zero and the search will degenerate into depth first search with looping (so it's also not complete).
9. No. After leaving the starting city, ignoring the constraints of the problem would go straight back to the starting city.
10. No. Greedy best-first search is not optimal and is incomplete. In addition, since this is a graph, it can cause a loop problem.
11. Yes. Each city is connected to another city.
12. Yes. Greedy best-first uses just the heuristic function f(n)=h(n). And here the heuristic is the distance to the closest city not yet visited. This way an optimal solution can be found.
13. Yes. If you can pick your start node. Simply look for the longest path and start at a node connected to it and you will skip it.
14. Yes. In this case greedy best first search degenerates to uniform cost which is guaranteed to find an optimal solution.

2. 10 points

In A* the nodes that have been generated but not yet expanded are sorted according to the value of f(n) = g(n) + h(n), i.e. the sum of the cost from the start to node n plus the estimated cost from n to goal using admissible heuristic h(.). Nodes that have the same f(.) value are ordered arbitrarily. Are there any domain independent criteria that could be used to re-order nodes with the same f(.) value? Explain your answer.

Correct Answer

Yes, arrange them by g(.) value, descending. If two nodes have an f(.) cost of 100 but one 80% g(.) and 20% h(.), it means most of that cost is certain (it's actual, not estimated, cost) and that we've almost reached the goal. If it's 20% g(.) and 80% h(.), it means that most of the cost is an estimate, and the estimate is probably an underestimate, so the costs are more likely to go up.

Scoring

0 points off if you gave an ordering scheme that was completely worthless. OK, perhaps we should have asked if there were any *useful* domain independent criteria. Since we didn't, if you gave at least one possible ordering criteria and explained it, you got credit.

3 points off if you gave an answer that showed a lack of understanding of search problems.

8 points off if you said it couldn't be done. 10 points if you said yes or no but gave no explanation.

Common Answers

1. Yes. Arrange by number of children. Would help optimize depth first search.
2. Yes. Arrange by f cost of children.
3. Yes. Sort by cost from start to them. Lower is better.
4. Yes. Arrange by g value. Prefer the one with highest h value because those are underestimates.
5. Yes. Sort on increasing g.
6. Yes. Sort on g. Whether increasing or decreasing depends on the heuristic. If g>h, the path length is probably more accurate because it's mostly actual cost, not estimated.
7. Yes. First found to last found. It wouldn't necessarily help with performance but it is a reordering criteria.
8. Yes. By their depth in the search tree, from highest to lowest / lowest to highest.
9. Yes. A node which has been reached with fewer prior states would be good in terms of memory. So nodes could be ordered by path length.
10. Yes. Order according to number of parents or how many nodes it takes to get to n. This is so that nodes with less parents would be visited first, reducing the number of nodes you have to expand and saving memory/storage space.
11. Yes. In route finding, it may be possible that two routes have the same cost but one route can be considered easier (fewer turns, use highways).
12. Yes, if reordering f will somehow save time/memory then it would be advantageous. There might also be an advantage to reorder based on how much of a role heuristic function played in g(n)+h(n) if there is a greater possibility for error in h(n) than g(n).
13. Yes. Because the have the same value, it's not entirely wrong to simply use a random function.
14. Yes. Order the nodes by f cost (low to high).
15. The algorithm might be slightly more resource efficient by selecting the path with the fewest nodes.
16. Yes. Order by the length of the path from the start node to the current node ascending.
17. Yes. A tie-breaker heuristic would need to be implemented estimating the minimum distance of the remaining tour to the goal.
18. Yes. Alphabetically or numerically (if nodes are named as numbers).
19. Yes. If we change the value of f(n), for instance h(n)=-2g(m), we can reorder nodes because in this case it works like depth-first search if each step cost is equal.
20. Yes. Time they are generated, newest nodes first. We can reduce the number of nodes searched by looking at deeper nodes because they are more likely to be near a solution.
21. No.
22. No. We don't know the heuristic, we don't know if it's a minimization or maximization problem. We don't know how precisely what cost means and whether higher or lower cost is better.
23. No. If there was you'd just have a better heuristic.
24. Since A* is admissible, you could order nodes but it's unnecessary.
25. No. Nothing will improve the solution as equivalent costs to the same state are equivalent in terms of the performance measure.

3. 15 points
You are given a state-space search problem, i.e. you are given the initial state, the operators, and a goal state. You come up with your own heuristics, apply A*, and produce an optimal solution.

3.1. What properties should your heuristics satisfy to guarantee the solution is optimal?

Correct Answer
The heuristic must be admissible, which means it cannot overestimate the distance to the goal. It the search space is a graph, there is potentially more than one path to a node, some less expensive than others. It is helpful for the heuristic to be consistent (estimated costs don't go down more than the amount it took you to get to that node) since that guarantees A* will find the least expensive path to that node first. If the heuristic is not consistent, you can still get an optimal answer but it requires A* to recheck nodes that are on the closed list.

Scoring
This part of the question was worth 8 points. To get full credit, you needed to explain what admissible meant. Technically, admissible means that it should either underestimate *or give the true cost* but people in the field tend to use "underestimate" here to mean "underestimates or matches" so no points were taken off for saying underestimates. Consistency for graph spaces was not required. Some students pointed out that the heuristic should return a distance of 0 when it was at the goal, which is true but not required here for credit.

5 points was taken off for not explaining what admissible meant or explaining it badly.

7 points was taken off for being completely wrong (you were given one point for making a good faith effort).

Common Answers
1. It should be admissible, i.e., it must *over*estimate *actual* cost.
2. It should be admissible, i.e., an underestimate
3. h(G)=0. It should be admissible and consistent. Consistent implies admissibility.
4. A* search using TREE-SEARCH is optimal if h(n) is admissible. Using GRAPH-SEARCH, it's optimal if h(n) is consistent.
5. It is consistent, meaning n has a successor n' reached by a, h(n)<h(n')+cost of moving to n'.
6. To ensure optimality, the heuristics should result in the fewest possible states visited. Your heuristic should explore all possible options before deciding how to act.
7. It should prove, at each step in the search tree, that a path to an optimal solution exists and necessarily use the heuristic to find it.
8. It must be admissible, or never overestimate cost. It must also be consistent.
9. If the space might be a graph, h() must be both admissible (never overestimates) and consistent ($h(n) \leq c(n,a,n')+h(n')$).
10. $h(.) \geq 0$. $h(.) \leq c(.)$ (estimated cost to goal smaller than actual cost). h(goal)=0.
11. There is no such heuristic, the point of a heuristic is that it is not an algorithm and just finds probably solutions.
12. You cannot expand f_{i++} until f_i is finished. A* expands no node with $f(n)>c^*$.

3.2. Is the optimal solution unique? If yes, explain why. If not, describe as many conditions you can think of that could cause finding multiple solutions.

Correct Answer
No, there could be multiple optimal answers because there could be multiple paths whose costs are equal. As an example, consider being downtown trying to get to the opposite (diagonal) corner. You can go right and up or up and left. Barring an M.C. Escher world, the two paths should have the same cost.

Which "best" path is found might dependt on which heuristic you use or on how you break ties.

Scoring
This question was worth 7 points. 5 points was taken off if you gave an incorrect answer. 7 points were taken off if you answered "yes" or "no" with no explanation.

Common Answers

1. No. (*draws diagrams to show; i like diagrams*).
2. No. Can have ties (multiple equal f costs).
3. No. If h(n)=0, the search becomes uniform cost and uniform cost searches can have multiple optimal paths.
4. No. It could be a full and balanced tree with unit cost for each edge.
5. It depends. There might be possibly an infinite number of nodes that satisfy the properties of completeness and optimality.
6. Yes.
7. Yes. h(n) is consistent so A* will explore all paths. When it finds a solution, it is unique because A* looked at all the other possibilities.
8. No. Could have multiple ways of reaching the goal. Certain domains could cause multiple solutions.
9. Yes. Unless it's a graph, where other optimal solutions may not have yet been expanded. This would be the case when the cost of reaching the goal is same on two paths but A* selects one of them randomly.

4. 20 points

Imagine a futuristic world where numerous types of robots are used. One is an armored transport vehicle whose job is to take a person to a location of that person's choosing. The transport only drives to approved locations and only on special roads dedicated to transport robots. The speed limit is the same for all roads. Traffic is not an issue as all roads are single lane in each direction and all traffic moves at the same speed. There are no issues related to sensing. The robot software is implemented using a goal-based architecture. The following is the PEAS description:
Performance: Get person to location, take shortest route
Environment: Client, enemies, other vehicles, roads
Actuators: Steering, accelerator, brake, speaker
Sensors: Camera, sonar, speedometer, GPS, voice input

4.1. Describe the environment. Is it fully or partially observable? Deterministic or stochastic? Static or dynamic? Discrete or continuous? Single or multi-agent? If multi-agent, who are the other agents and are the agent interactions competitive or cooperative?

Correct Answer
Partially Observable because of enemies and because it's the real world.
Stochastic because of enemies and because it's the real world.
Dynamic because of enemies and because it's the real world.
Continuous because it's the real world.
Multi-agent competitive because of enemies and cooperative because of passenger/client (NOT because of other vehicles because there is no intended/designed interaction with them; if we wrote the code for this agent, we would not need to write anything to interact with the other friendly vehicles unless you believe there would be potential problems at intersections).

This question is a simplified version of the taxi cab example in the book. As both the book and the HWI answer key mentioned, the real world is almost always partially observable, stochastic, dynamic and continuous. Fully observable, deterministic, static, discrete environments or environments close to these are normally abstract (e.g., nine queens problem), simulated (e.g., chess) or specialized (assembly line robots). Note that, despite how some people interpreted the text, dynamic and stochastic environments do not necessarily mean the environment is partially observable (e.g., the agent cannot see all the data *it cares about*). Examples include many assembly line robots and exploratory agents modeled on insect behavior.

Scoring
There were five(ish) answers here so they were one point each. No partial credit. Answers that differed from the preferred answer but were backed up by a good, detailed, unambiguous explanation were given credit.

The most common errors were forgetting one of the criteria (normally Dynamic/Static) and stating the environment was fully observable or static. Several students also said the environment was strategic despite that not being an option. Several students also said Episodic despite the question not asking about that (if it had, episodic would be the wrong answer).

Common Answers

1. Partial (because of enemies). Stochastic (because of weather, enemies, conditions). Dynamic (because not given time to think while being shot at by enemies, running over things or crashing). Continuous. Multi-agent competitive (enemies) and cooperative (client and

other vehicles).

2. Partial. Dynamic. Continuous. Strategic.
3. Partial. Stochastic (meteor could hit). Dynamic (meteor). Discrete (actions are a finite number of distinct states). Multi-agent.
4. Full. Stochastic. Static. Continuous. Single-agent. (*no explanations*)
5. Full (if not worry about other robots; problem states no traffic issues). If robots can fail or person can talk to the robot en-route, it's Partial. Stochastic (because of equipment failures). Dynamic. Continuous. Multi-agent cooperative (because interacts with user).
6. Full (it gets all the info it needs from sensors). Dynamic (other robots on road, enemies not predictable). Stochastic and continuous (agent continuously acting until goal reached, it never stops acting). Multi-agent competitive (enemies) and cooperative (other transport vehicles).

4.2. What in this problem makes a goal-based architecture a good choice?
Why aren't the simple or model-based reflex architectures a good choice?

Correct Answer

Goal-based is a good choice here because the agent has a goal (get to a specified destination) and must make a plan to get there (for example, go straight five blocks, go left two blocks then turn right). Reflex is a bad choice because reflex agents don't understand goals and can't make plans. They simply react to something they sense. For example, they can have the condition-action rule "if there is an intersection, take a left" or even "if it's the intersection of 8th and E Street, take a left" (although a table of actions like this might quickly grow too large to fit into memory) but not "if my goal is Cradock Marine Bank and I'm at 8th and E Street, take a left".

Scoring

The correct answer should both explain why the goal-based architecture is good and the reflex one is bad. No points were taken off if an explanation for only one was given if it made it obvious how it differed from the alternative (e.g., "goal-based is better because the agent needs to make a plan" with the clear implication that the reflex agent cannot).

2 points were taken off if the answer said something generically negative about reflex agents that were not specific to this question. For example, "reflex agents cannot handle partially observable environments" does not explain why a reflex-driven vehicle couldn't handle this environment. "Reflex can't handle enemies" is specific to the question but doesn't explain why exactly that might be true (for the record, fighting or avoidance is one of the things a reflex agent can often do well).

2 points were taken off if you said a reflexive agent can't handle dynamic environments. Reflexive agents are actually quite good in dynamic environments, so long as a rule exists for the encountered situation (e.g., road blocked). Goal-based agents have no advantage here since it must recognize when there is a problem (e.g., blocked road) and know how to solve it (e.g., back up to the last intersection then generate a new route).

3 points were also taken off if you said that a reflex-driven agent would get you to your destination but would not necessarily take the shortest route. There is no reason to believe that a reflex-based agent would ever get you to your destination. If it did, there's no reason to believe it would stop and let the passenger out (remember, reflex-agents have no mental conception of goals, just hard-wired behavior).

4 points were taken off if you did not show that you understood the details of each agent architecture. For example, if you just said that reflex-based agents are dumb or goal-based agents are good or if you gave an explanation that simply wasn't true (e.g., goal-based agents can learn even when they're not learning agents). Confusing a customer's goal (e.g., the customer is trying to get some place) and the technical details (e.g., goal-based agents can select from options because they have a mental representation of a goal) is included in this category.

Common Answers

1. Reflex bad because the environment is partially observable.
2. Reflex bad because, without goals, the robot does not know where to head at a crossroads.
3. Model-based Reflex bad because environment is fully observable and single agent. Simple Reflex bad because they have limited intelligence and ignore percept history.
4. Reflex bad because it can't deal with enemies. Also, the best path is dynamic and requires planning. Model-based can't remember enemy positions because they are dynamic.
5. Reflex bad because they will get you there but not necessarily on the shortest path, leaving the passenger more at risk for a longer amount of time. They react to danger, not avoid it.
6. Reflex bad because it's just condition-action rules, might take a longer route or not get to the destination at all. Does not have a goal.

7. Reflex bad because the world is too dynamic. For example, car/robot breaks down and blocks road. Reflex agent would be dumb founded.
8. Model-based Reflex bad. Much of the world is predictable most of the time. There isn't much to react to.
9. Reflex bad because, since the performance measure includes taking the shortest route, a simpler reflex-architecture will not be able to guarantee the shortest route using condition-action rules without an expansive table.

1. Goal good because it can modify the experience which helps construct the AI's choices.
2. There is a goal (location to get to) which allows the agent to pick the most desirable action.
3. Need goal to describe which situations are desirable (e.g., being at the passenger's destination).
4. Goal-based agents can plan. Can take future possibilities into account.
5. Goal-based because getting person to location is better when there are enemies around compared to non-goal based, where the agent would be more predictable.
6. Easy to search. Easy to get initial and goal states and predict cost.
7. Goal-based agents can think about the future and learn over time.
8. Goal-based architecture is necessary because if the customers don't reach the goal they don't care about how successful the robot was.

4.3. What could you change in the problem to make it suitable for a simple or model-based reflex agent? (Be specific; it is OK to change the type and purpose of the robot).

Correct Answer

Reflexive agents cannot pick between choices such as "go left" or "go right". Their behavior is hard coded to always do one action in one context. To make the situation one where a reflexive agent would do well, remove choices from the environment. For example, make the routes only point to point on straight roads or make the transport follow a set route like a bus. You could also change the purpose of the robot. For example, instead of being a military transport, it could be a military sentry that looked for enemies and fired if it saw one. Or it could be the defensive or turret control system on an armored transport (different responsibilities in a transport could be handled by different agents).

For the most part, students did well on this question.

Scoring

3 points were taken off for the answer "don't worry about taking the shortest route" since it is not clear that a reflexive agent could find any route and because reflexive agents wouldn't recognize the correct destination even when it got there.

3 points were also taken off if you suggested making the environment fully observable without giving an example of how it could be changed and what the impact would be. More specifically, since the core problem is that the reflexive agent cannot make plans and thus cannot find a route, it is unclear how making the environment fully observable would help.

4 points were taken off for any other answer that did not show an understanding of reflexive agents, such as suggesting that giving the agent more work to do (e.g., find the best route on a busy road while fighting enemies)

Common Answers

1. Make environment fully observable. Give it a network sensor array.
2. All places directly connected, no crossroads.
3. Make the environment partially observable, dynamic and multi-agent (where traffic and sensing are issues). For simple reflex, make the decision be based only on current location and situation.
4. Make the robot fly, remove the other agents. Then the robot could just point at the destination and make corrections as detected.
5. If they have weapons, have avoidance models built in to make the robot try to avoid any enemies seen. (*not sure why you'd need weapons to run away from enemies*).
6. Remove shortest route from the performance criteria / Change performance measure from shortest route to any route.
7. Road or robot only takes you one place (aka train).
8. The robot should be designed to securely transport people. It'd need to react to things like gunfire or explosions. It'd need to speed off as quick as possible, modify speed and maybe route.
9. Simplifying the route structure (i.e., a circular path, one-direction).
10. If only needed to keep client alive, could simply just react to attack.
11. Make it so that the roads are binary, i.e., so that an internal state is not necessary.

12. Tale customers to random locations, not to the place they want to go.
13. Robot's purpose is to detect how close an enemy is. The agent could just stand looking for enemy vehicles.
14. Only one robot.

4.4. Under what circumstances would a utility-based agent be appropriate?

Correct Answer
A goal-based agent can say whether something does or does not help it achieve a goal but it cannot say that one useful action is better than another. A utility-based agent would be needed if one option is preferred over another. Common answers here were for the transport to balance some mixture of time, distance, fuel consumption, safety and comfort. The answer needed to show that the agent (the transport) was making a decision between two good (but not equally good) choices.

As with the last question, most students did well on this one, although some students should have been a little more specific and detailed in their examples.

Scoring
1 point was taken off if your example was not very clear or slightly off in relation to this question.

2 points were taken off if no example was given (e.g., saying that "it should pick the most desirable route" without explaining what criteria might make one route more desirable than another).

3 points were taken off for "try to keep the client safe". This is not specific enough. What about a utility agent, vs. a goal-based agent, makes them better at keeping someone safe? Because the goal of the question is to make sure you understand the specifics of the different conceptual agent architectures, answers had to have enough detail to show your grasp of the architectures.

4 points was taken off if the answer was completely wrong (since the question is five points, in essence, one point was given for at least trying to answer the question).

Common Answers
1. Need a gradient of satisfaction. Some rating of safety on each route.
2. Worry about fuel consumption.
3. Need to worry about customer satisfaction. For example, if traffic conditions vary by road, it should choose the less overwhelmed road to optimize arrival time.
4. Enemies play a larger role, have to balance safety vs. shortest route vs. speed.
5. No enemies, just a simple transport mission.
6. A notion of "happiness", for example, time – the faster the happier – or safe.
7. If we are trying to maximize profits. This will allow agent to maximize "happiness" of the agent so that it can be very much performance based instead of goal based. Ex. – the agent would be better off carrying 1 person for $500 than 50 people for $1 each.
8. If certain routes were more desirable than others. You could then pick the most desirable path.
9. If the robot played a role in making its passengers comfortable and passenger comfort levels sufficiently high to be similar to the happiness gained from arriving at the destination.
10. Can go off road when in danger. Could be used to find the most cover from an enemy.
11. Doing multiple tasks such as sensing the nearest enemy and transporting people.
12. Two-way, multilane roads with different speed limits. Then the robot must balance safety and speed.

5. 10 points
Write in Lisp a function that takes two lists and returns a list constructed by interleaving elements of the first list with elements of the second list, so that the resulting list contains one element from the first list, one from the second, etc. in this order. Terminate the process when one list ends by including the rest of the other list in the result. It should work like this:
(interleave '(a b c) '(1 2 3)) ==> (A 1 B 2 C 3)

(interleave '(a b c) '(1)) ==> (A 1 B C)
(interleave '(a) '(1 2 3)) ==> (A 1 2 3)
(interleave nil '(1 2 3)) ==> (1 2 3)

```
;; this makes recursive calls alternating the lists
(defun interleave (lst1 lst2)
  (cond ((null lst1) lst2)
        (t (cons (car lst1) (interleave lst2 (cdr lst1))))))

;; this is more cumbersome
(defun interleave (lst1 lst2)
  (cond ((null lst1) lst2)
        ((null lst2) lst1)
        (t (cons (car lst1)
             (cons (car lst2) (interleave (cdr lst1) (cdr lst2)))))))
```