6.034 Artificial Intelligence (Spring 05)

Formulating Search Problems

Due: Wed. March 2 11am, *before* Lecture

1 Instructions

In this assignment we ask you to look at problems drawn (but simplified) from real research or application problems. The assignment is to think very carefully about how you would solve one of these problems using the representations and algorithms covered in Chapters 2 and 3. The problems are specified very loosely: part of the problem is for you to make (and justify) reasonable assumptions about what's important. However, if you have basic questions about what is meant by the problem statement, you can send the staff **one** e-mail *before midnight on Monday February 28*, asking for specific clarification. Also, the recitation meetings on Friday will be devoted to answering your questions.

1.1 Hand-in

You are to hand in a 3-5 page paper (1500-2500 words) by Wed. March 2 at 11am. We have attached a sample solution to a simple problem. This is not necessarily the best possible solution to the indicated problem, but it illustrates what we are looking for.

Bring a print-out of your paper and turn it in before lecture on March 2. If it is late, it will be subject to a 20% per day late penalty, up until your Friday recitation, after which it will be worth 0 credit.

During the following Friday (March 4) recitation, we will give you a 10 minute appointment to discuss your paper with one of the staff. You should be ready to give a concise 5 minute verbal presentation of what you did and answer questions for another 5 minutes. The presentation and QA will factor into your grade for the project.

You may have discussions with your classmates to better understand the problems and you may look up references. You must write the paper completely independently and be able to justify all aspects of your formulation at the oral presentation.

1.2 Paper outline

Your paper should contain the following sections, in the order indicated below. In all cases, justify your choices. In the "Problem Definition" and "Algorithm" sections, indicate what alternatives you considered and ruled out and why you made the choices you did.

1. Background

   Provide a brief textual description of the problem. Elaborate on the given problem statement, providing some more detail about the particular class of problems you chose to solve.

2. Problem definition

   - Define the type of search problem, e.g. path search, constraint satisfaction, optimization.
   - Indicate precisely what data is needed to solve the problem.

- Define precisely the states (or variables/values), descendants, cost function, start state, goal test as appropriate.

3. Complexity

   - Estimate the size of the state space for realistic problems as a function of the parameters specified in the problem. Estimate depth to goal and branching factor. Say whether each of these things grows linearly, polynomially, exponentially, or worse, as a function of the parameters.

4. Algorithm

   - Recommend the most efficient algorithm that's guaranteed to find an optimal solution (a satisfying assignment in the case of a CSP) from among the ones we have studied. Specify any implementation issues that are specific to this problem. For example, if a heuristic is needed, specify it in detail here. Discuss the use of expanded or visited lists.
   - Now specify an algorithm that is not necessarily guaranteed to find an optimal solution, but that will be practical in the case of very large problems. In some of the problems we have given particular hints about how to develop an approximate algorithm; otherwise, consider local and greedy methods we have discussed in lecture.

5. Worked example

   Work out a small example by hand. Simplify as necessary to convey the key ideas in your approach.

6. Summary

   Summarize your conclusions. How feasible is solving the problem? Given a hard time constraint in advance (an hour, say) will you always find *some* answer? What was hard about this problem? What required insight about the specific problem?

7. References

   If you consulted any references, list them here.

1.3 Grading Criteria

Remember that this assignment will account for 7.5% of your grade, so it is important. We will break down the grading as follows:

- 60% Technical formulation
- 20% Clarity of writing and explanation
- 20% Oral presentation

The problems are marked with difficulty levels of 1 or 2. If you solve a level 1 problem, the maximum score for the technical formulation will be 55%; for level 2 problems, the maximum score is 60%.

These problems don't have a single right answer. We don't necessarily know the best way to approach them, or even have a particular concrete approach in mind. It will take some thinking to come up with a good answer. Start early so you can mull the problem over and so you can edit your text at least once.

Be sure to clearly explain your reasoning and justify your decisions.

2 Problem Descriptions

Below are some problem descriptions for you to choose from. If you have some alternative problem that you are interested in, please e-mail a description to the staff by midnight on Thursday February 24, and we will let you know if it's possible to substitute that.

2.1 Robot Motion Planning

In robotics, one of the classic problems is figuring out how a robot should move in an environment so as to move from its current position to some target position while, for example, avoiding collisions and/or minimizing time. Here are a couple of examples of this type of problem:

2.1.1 Large spaces (Level 2)

Consider planning the path for an "off-road" robot vehicle in a very large space (map of continental US or a map of a large chunk of Mars) described as a grid with each grid square being 1 meter on a side. We assume that the robot is small relative to the size of the grid squares, that is, it fits entirely within a grid square and takes some time to traverse a grid square. Each grid square is marked by a value between 0 and 1, indicating how long it takes to move through that space or -1 indicating that it is impassable. A "typical" path might need to be thousands of kilometers. Discuss the complexity of the problem in terms of the area of the map.

One strategy for developing an approximation method would be to solve the problem hierarchically, starting with a very coarse discretization of the map and gradually refining it.

2.1.2 Robot arm (Level 2)

Consider a planar jointed (robot) arm (such as shown in figure 1) with at least six joints. You can control the robot arm by specifying angles for all the joints; the base of the robot is fixed. You are given a complete description of the environment as a set of polygons that represent obstacles. The problem is how to move the robot arm quickly from its current position to some target position (specified by its joint angles) without colliding with any obstacles. Assume that you are given a function that can test whether a position (joint angles) of the robot arm would lead to a collision. You need not worry whether the robot is in collision with itself. Discuss the complexity of the problem in terms of the number of joints and the granularity of discretization of the joint variables.

The hierarchical decomposition approach outlined in the previous problem might be less useful here. Explain why, using complexity arguments.

Another strategy for approximation in this domain is to randomly generate arm configurations in the free space and use them as the states. Assume that you have a function that can test whether the straight-line (in joint space) path between two points can be traversed without collision. Outline

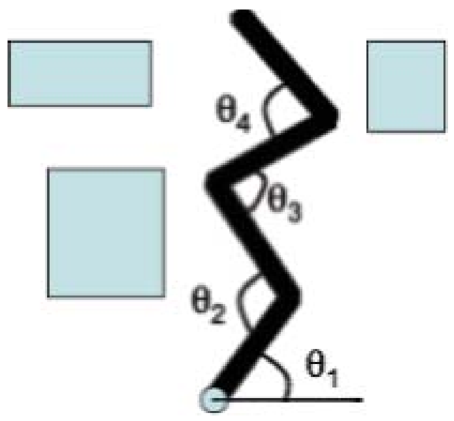

Figure 1: A planar robot arm.

a search algorithm based on these ideas and explain (informally; doing this formally is *hard*) what properties of a particular set of obstacles would make it easier or harder to solve.

2.2 Planning

More generally, we are interested in planning sequences of actions required to achieve goals.

2.2.1 Deliveries (Level 1)

Plan the delivery schedule for a fleet of K trucks and N packages in a city with M destinations with each destination directly connected to some other destinations (each labeled with distance). Each truck can carry a fixed number of packages. All the trucks start at the depot in the morning and the goal is to deliver all the packages and have all the trucks end up at the depot. Ideally we want to minimize the total distance traveled by the trucks. Discuss the complexity of the problem in terms of K, N, and M.

Initially, assume that the assignment of packages to trucks is given; explain how you would solve the problem. Now, give some ideas how you might assign the packages to the trucks to reduce the total distance covered by all the trucks; can you do it optimally in reasonable time?

2.2.2 Cooking (Level 2)

Plan to cook a large meal on a K-burner stove. You have to cook N dishes, each of which must be started in a clean pan, and needs to cook for a certain (uninterrupted) amount of time (which varies among dishes). You have a limited number (typically less than N) of pans. If you need to re-use a pan, it will take time to clean it. The dishes must all be cooked, and you would like for them all to be as hot as possible when you serve them (once a dish has finished cooking, it begins to cool). Discuss the complexity of the problem in terms of N and K.

There seem to be two ways of representing time in this problem: one is to discretize it uniformly; another is to consider a “step” of the plan to be whenever a dish is finished cooking. Please discuss the relative advantages and disadvantages of these representations.

2.3 Scheduling

2.3.1 Airline Personnel (Level 1)

Given a set of N flights, each of which requires a set of personnel with particular skills (pilot, co-pilot, head flight attendant, etc.) and a set of P personnel that require assignments (one of which can be office work), find a good assignment of personnel to roles on flights. There will be many conditions that a schedule has to satisfy. Here are some of them, but you should add at least three more that seem reasonable:

- A crew member has to be located in the departure city of a flight at least one hour before the flight is scheduled to take off.
- Pilots can only fly planes on which they're certified.
- Crew members can only work 8 consecutive hours without a rest.

What if, in an attempt to improve employee satisfaction, we decide to allow people to express preferences about who they work with? Would that make the problem any more computationally difficult? Show how you would have to change your formulation (if you would) in order to deal with it. Discuss the complexity of the problem in terms of N and P.

2.4 Structural Biology

Most of the chemical reactions that are necessary for life are mediated by proteins, complex (approximately) linear molecules that fold up into characteristic shapes in three-dimensions. These shapes are ideally matched to fit the shapes of other molecules, not unlike a lock is matched to a key. Part of understanding the behavior of proteins and designing new drugs is determining the folded shape of actual proteins. This can be done via NMR (nuclear magnetic resonance) and via X-ray diffraction. Here are a couple of computational problems that arise in that context.

As a **very** simplified model of proteins, assume that a protein is a **sequence** of N spheres of radius 1 unit. Real proteins are made up of a sequence of different amino acids with different physical and chemical properties; here we will use the same model for all of them, so the particular sequence is irrelevant.

The distance between adjacent sphere centers is approximately 2. The angle between three adjacent spheres is in the range of 90 to 135 degrees. Note that no two spheres are allowed to collide.

2.4.1 NMR distances (Level 2)

One can use Nuclear Magnetic Resonance (NMR) to detect the approximate distances between some spheres in a protein chain. Assume that you are given a few (many fewer than N^2) measured distance ranges of the form $lo_{i,j} \leq Dist(i,j) \leq hi_{i,j}$ where i and j are the indices of spheres in the protein chain. Given this data, find positions for one arrangement of all N spheres in three dimensions that satisfy the constraints between adjacent spheres as well as the measured distance range constraints between non-adjacent spheres. Typical proteins are sequences of anywhere from 60 to thousands of these spheres. Discuss the complexity of this problem as a function of N.

Note that, since the distance between adjacent spheres on the sequence is fixed, the angle between three adjacent spheres can be described by the distance between the first and the third.

2.4.2 Xray density (Level 2)

Through X-ray diffraction of protein crystals, it is possible to obtain a three-dimensional array of densities (between 0 and 1), each indicating the probability that there is a sphere of the protein at that location. This data is quite noisy. Assume that each entry in the array covers a volume that is 0.5 units on each side. The goal is to find positions in the array for the N spheres so that the product of the probabilities of those positions is maximal. (Hint: use the log of the product of probabilities as your score). Of course, we have to respect the basic distances and angles among the spheres we indicated in our definition of protein structures, and make sure that the spheres don't collide. Discuss the complexity of this problem as a function of N.

3 Sample solution

The following text consists of a partial solution, plus some hints and questions (in italics) to help you in developing your own solution.

3.1 Background

The problem considered here is that of an automated mini-bus that has to plan its path through a city to pick-up and drop-off a set of N passengers that have called in. The passengers can be picked up and dropped off in any order. The mini-bus can fit K passengers; it may be given more than this number of passengers to deal with during a trip. After all the pick-ups and drop-offs, the bus then returns to the depot. The bus has a limited amount of gas when it starts and will need to refuel as necessary during its trips. The cost of gas varies at different gas stations throughout the city; there is also cheap gas at the depot. The objective is to do the trip quickly without excessive cost for gas.

3.2 Problem definition

This is clearly a path-finding problem. The objective is to find a path through a graph that represents places in the city that takes all the passengers to their destinations while minimizing some combination of time and gas costs.

- Data: A graph representing the city.
  - A node in the graph represents an intersection of streets or highways; call this an *i-node*. We assume we have (x,y) positions for i-nodes, so we can compute straight-line times between them (assuming fastest travel). Note that we will also need a database of addresses that will tell us the nearest i-node for each passenger origin and destination.

    We'll ignore the problem of going from the nearest intersection to the actual address, except that the i-node for an address must allow reaching the address, for example, in one-way streets, the i-node allows entering the street where the address is. This is done before the search actually starts.

    We need a list of i-nodes with gas stations and the price of gas at each gas station.
  - A link represents a segment of a street or highway; call this an *i-link*. The links are directed, so that we can model one-way streets. Two way streets are represented by two separate links. Associated with each link is an estimated time to traverse it as well as the estimated gas needed to traverse the link.
- States: A search state is `(i-node gas-in-tank in-bus-dest-i-nodes pick-up-origin-i-nodes pick-up-dest-i-nodes)` We need to know the destinations of the passengers in the bus and the origin and destination of passengers we have yet to pickup. Note that we don't need to remember anything about passengers we have dropped off, nor the origins of the passengers currently in the bus, since that doesn't affect our future choices.
- Descendants: There are 4 action types that define descendant states. Assume that the current state is defined as above.

  - Drive to adjacent intersection (`i-node2`): For all `i-node2` connected to the current `i-node` and not on the current path. Can only be done if the gas consumption along the link to `i-node2` is less than the current `gas` level in the bus. The result is changing the current `i-node` and decrementing the `gas`. The cost of the link to the new search node is the estimated time associated with the i-link from `i-node` to `i-node2`.
  - Pick up passenger: If the current `i-node` is one of those in `pick-up-origin-i-nodes`, and if the number of `in-bus-dest-i-nodes` is less than K, we add the corresponding entry in `pick-up-dest-i-nodes` to `in-bus-dest-i-nodes` and delete the entries in `pick-up-origin-i-nodes` and `pick-up-dest-i-nodes`. The link cost is 0.
  - Drop-off passenger: If the current `i-node` is present in `in-bus-dest-i-nodes`, we remove them. The link cost is 0.
  - Fill gas tank: If the current `i-node` has a gas station, the we can fill the tank. The link cost is $\alpha * \textit{gasBought} + \beta$. The idea is that filling the tank takes some time (β) and we will also penalize the path by some factor (α) that converts gallons to time. Changing this factor will change the tradeoff between time and gas consumption.

- Initial State: The initial state is `(depot-i-node initial-gas () all-origin-i-nodes all-dest-i-nodes)`.

- Goal Test: The goal is simply `(depot-i-node gas () () ())`, where `gas` needs to be greater than or equal to 0.

3.3 Complexity

The size of the state space grows linearly in N (the number of passengers) and in the number of intersections (which is itself likely to be linear in the area of city). The branching factor is a small constant (about 4 next intersections, plus pickup, dropoff, buy gas). The length of a single "trip" (dropping off a vanful of passengers) is likely to be *something*. And we expect the total path length to be *some number* of trips. So, the worst case number of states visited by an exact search is *something*.

3.4 Algorithms

3.4.1 Exact Algorithm

The natural algorithm to use is A^* since we want to minimize a path-cost measure and we would like to use a heuristic function to focus the search towards the goal. A uniform-cost algorithm would probably be too slow; it would end up expanding many nodes that make no progress towards the goal. The difficulty is in constructing an appropriate heuristic.

A trivial heuristic, which is probably not very good (but better than nothing) is to simply use the estimated time from the current i-node to any node in `in-bus-dest-i-nodes` or `pick-up-origin-i-nodes`.

We can base a better admissible heuristic on the following observation. The remainder of the path from `i-node` back to the depot will have three "legs":

1. from `i-node` to some node in `in-bus-dest-i-nodes` or `pick-up-origin-i-nodes`. Call it the input i-node.

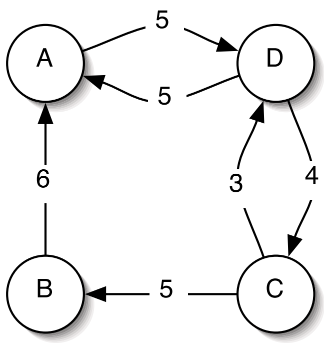

Figure 2: Road map for planning problem.

2. from some node in `in-bus-dest-i-nodes` or `pick-up-dest-i-nodes` (call it the exit i-node) back to the depot.

3. some tour of all of the i-nodes in the state, starting at the input i-node and ending at the exit i-node.

A (rough) lower-bound on the total remaining path length is: For every i-node in the state (and the current i-node), find which other i-node in the state (and the depot i-node) has the lowest lower-bound-time to it. Sum these minimum times for each i-node in the state (and the current i-node).

Since we have to pass each i-node in the state (and then go to the depot), the sum of the lower-bound-times to the closest i-node for each i-node that needs to be visited is a lower-bound on the actual path length. So, we have an admissible heuristic. We can keep around a table of i-node times so as to make computing this heuristic not too slow.

We should use an expanded list; we certainly don't want to re-expand states. The key question is whether we ever need to re-expand an i-node, that is, does the optimal path ever go through the same intersection but with a difference in the rest of the state entry? The answer is obviously yes, in that only the Drive action changes i-node; the others leave the i-node the same. But, beyond these trivial loops, you may need to backtrack to some intersection to pick up a passenger that did not fit in the bus the first time you went through that intersection.

3.4.2 Approximation Algorithm

Discussion of local search: How will you generate an initial trajectory? What moves will you consider between trajectories? What moves will you accept? How will you decide to terminate?

3.4.3 Worked example

Assume we have four i-nodes: `A`, `B`, `C`, and `D`. They are laid out in the figure 2. The arcs are labeled with the time to traverse, which, for simplicity is also the amount of gas consumed (in some strange units!).

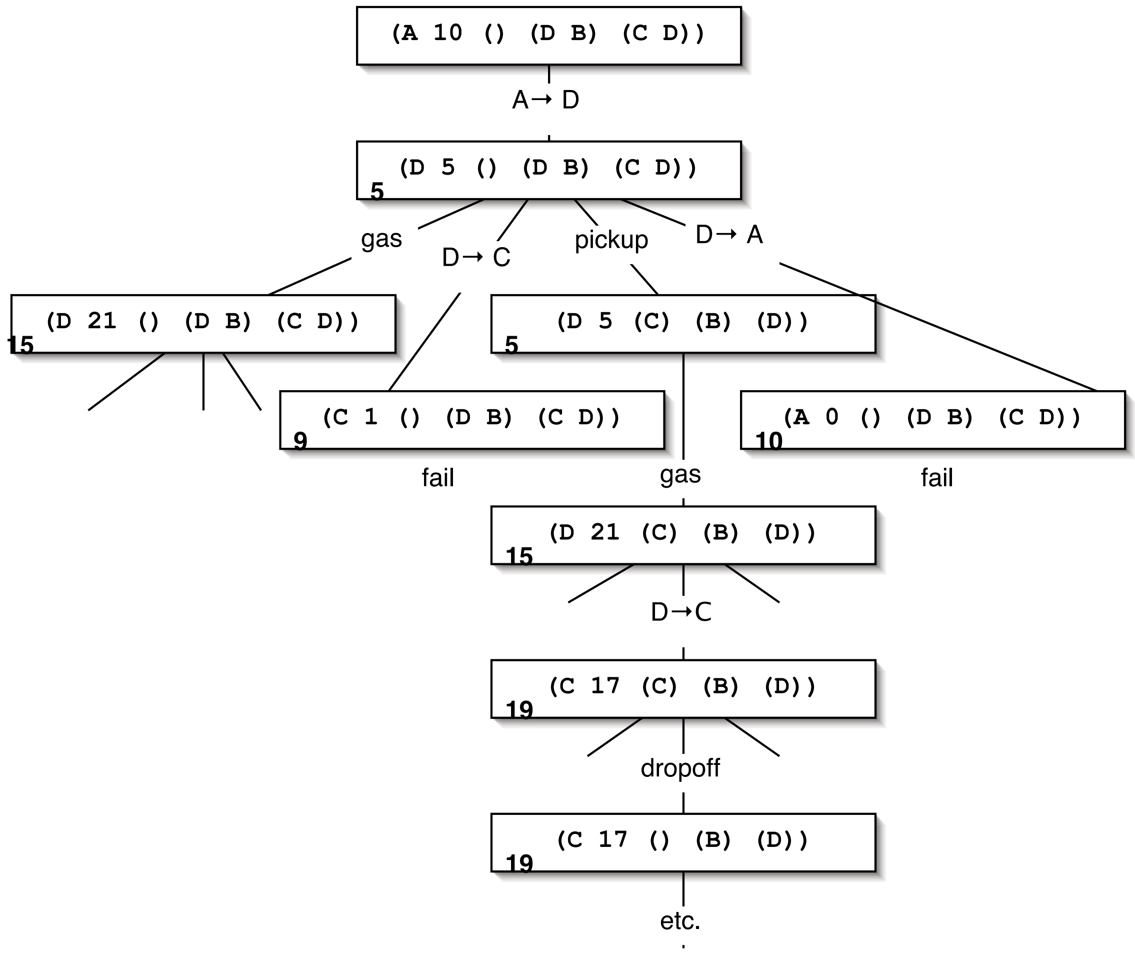

Figure 3: Partial search tree.

There is a gas station available at i-node `D`. The gas tank capacity is 21 units. At this station $\alpha = .5$ and $\beta = 2$. This minibus is actually a red two-seater convertible, so $k = 1$.

Let the initial state be `(A 10 () (D B) (C D))` that there is a passenger that wants to go from `D` to `C` and another that wants to go from `B` to `D`.

Figure 3 is a partial search tree. It shows all actions at level 1 and level 2, and then goes deeper down what is probably the optimal path to the goal. Note that when there is not enough gas left to traverse any arc, then there are no children left for a node. The past-cost-so-far is written inside each node in bold. *Your solution should also include heuristic values for each of the states in the tree.*

3.4.4 Summary

If our heuristic is good, the number of expansions is likely to be much better than than the number of i-nodes in a city, which is probably less that 100,000. Assuming the number of i-nodes is proportional to area of the city, we would expect an average trip to be on the order of the square root of the number of i-nodes.

The larger N will mean that more of the city will have to be explored. If N is much larger than K (say $K = 1$) there is likely to be lots of re-visiting parts of the city; this could make the worst-case be like doing N separate trips, which could get bad. A local search algorithm that gave up guarantee of optimality is almost certainly feasible.

The hard part of this problem is formulating a useful heuristic. The one here is a weak approximation to the original problem. It remains to be seen how good it would be in practice.

3.4.5 References

I did not consult any references.