Lecture 12: INFORMED SEARCH *

10-607 COMPUTATIONAL FOUNDATIONS FOR MACHINE LEARNING

1 Different Kinds of Search

Up until now, we have seen numerous examples of search problems, largely on graph and tree data structures. In this reading, we are going to focus on a specific search task: finding the shortest (weighted) path between two nodes in a graph. We have already introduced some algorithms for solving this problem e.g., breadth-first search for unweighted graphs and Dijkstra's algorithm for weighted graphs. These algorithms make no assumptions about what the "right" direction to explore is: they maintain a frontier of nodes that are equidistant from the start node and gradually add nodes to the frontier that are increasingly far away until they find the end node; crucially, these methods might have to explore nodes that are in the complete opposite direction of the goal. As an example, consider the following snapshots of Dijkstra's algorithm:

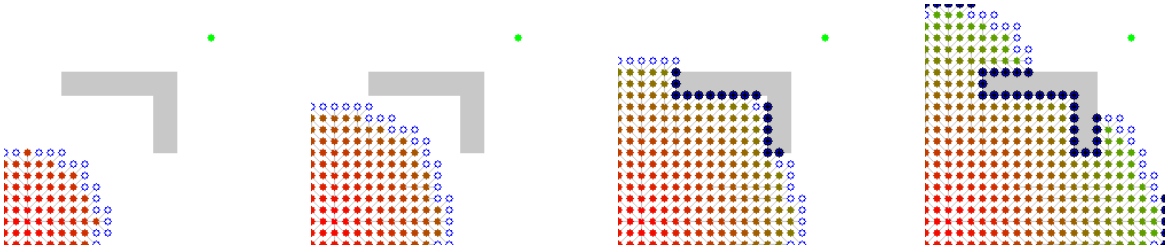

Figure 1: Dijsktra's algorithm for finding the shortest path: in this example, the grid can be thought of as an unweighted graph where each dot is adjacent to its eight immediate neighbors. Notice how the frontier (the open blue circles) expands outwards uniformly in all directions from the start node, in a roughly circular shape.

Source: https://en.wikipedia.org/wiki/File:Dijkstras_progress_animation.gif

These algorithms are examples of **uninformed** search algorithms: they don't make any assumptions about the relative positions of the start and end states. By contrast, **informed** search algorithms make use of a **heuristic** to guide their search. A heuristic h is some function that takes a node as input and returns an *estimate* of the distance between that node and the end state. as an example, if the goal were to find the shortest driving route between two cities, than a heuristic might be the distance between the two cities "as the crow flies" i.e., the distance of the most direct path possible between the two cities. Of course, this is just an estimate of the true driving distance between two cities as roads rarely follow the most direct path from one location to another. Note that heuristics are problem specific and not all problems permit a good or reliable heuristic (we will define desirable properties of heuristics shortly). As such, not all search problems can be framed as informed search; sometimes, we have to accept that we can not specify the general direction to search in and have to rely on uninformed search methods. However, if we can construct a good heuristic, then we can often dramatically improve the efficiency of our search. Consider the qualitative comparison between uninformed and informed search settings below:

*Compiled on Wednesday 29th November, 2023 at 19:19

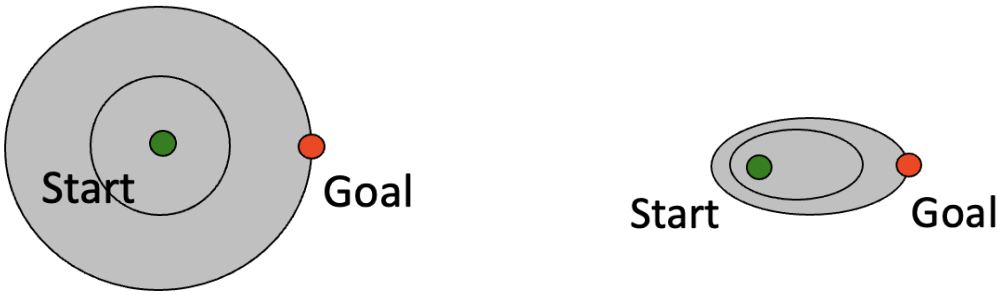

Figure 2: Uninformed (left) vs. Informed (right) search: in the uninformed setting, search algorithms need to expand the frontier in all directions uniformly. Informed search algorithms can leverage the heuristic to focus their search in the general direction of the end state (the "goal" in the figure) and (hopefully) find the optimal path much faster than uninformed search.

Source: http://ai.berkeley.edu/home.html

Let's consider two ways that we can make use of these heuristic functions for informed search.

2 Greedy Search

The first approach is perhaps the simplest: in each iteration, **greedy search** selects the frontier node with the lowest heuristic score for expansion. Practically, the frontier for greedy search can be implemented as a priority queue, where each node n's priority is $h(n)$; we dequeue the node from the priority queue with the lowest priority and enqueue that node's neighbors, again using h to assign priorities to all the new nodes.

Unfortunately, although greedy search is simple and straightforward, it is not guaranteed to be optimal; intuitively, greedy search can behave like depth-first search in an uninformed setting: it can blindly go down some path, following decreasing heuristic scores but ultimately missing the optimal path that had say slightly higher heuristic scores but lower true distances. Even worse, greedy search is not even guaranteed to find *a* path from the start to the end as it can get stuck in infinite loops. Of course, this suboptimal behavior can be exacerbated by bad heuristic functions i.e., heuristics that don't reflect the true relative distance between nodes and the end node. Consider the following example:

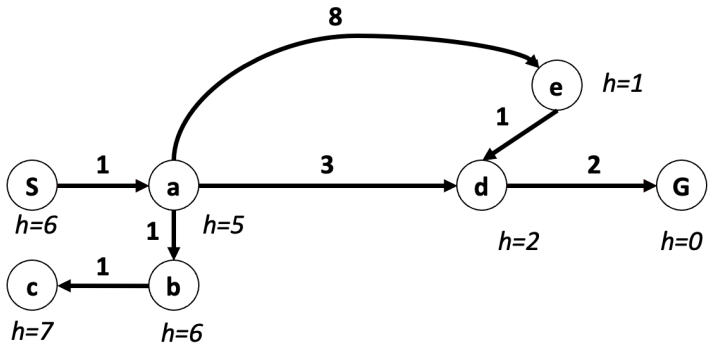

Figure 3: An example weighted graph: the start state is the node labelled "S" and the end state is the node labelled "G". The number next to each edge is the weight and the heuristic score for each node is shown next to the corresponding node.

Source: http://ai.berkeley.edu/home.html

In the example above, greedy search would first add node a (the only neighbor of the start state). From there, greedy search would expand node a and add its neighbors: b, d and e. Then, because e has the lowest heuristic score ($h(e) = 1$), greedy search would expand e, followed by d and then finally the goal state, G. The resulting path length is $1 + 8 + 1 + 2 = 12$ as compared to the true optimal path length of $1 + 3 + 2 = 6$ (S $\rightarrow$ a $\rightarrow$ d $\rightarrow$ G).

3 A* Search

A better informed search algorithm is known as **A* search** (pronounced "A star search"). Like greedy search, it also maintains a frontier using a priority queue. However, the priority queue for A* search computes priorities by combining the heuristic with the distance from the start state (the *backward* cost). Formally, the priority for node n used by A* search is

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

where $g(n)$ is the shortest distance from the start state to node n and $h(n)$ is the heuristic score of n. Intuitively, this priority is an estimated shortest distance from start to end if we were to go through some node n where the estimate relies on the heuristic to approximate the unknown portion of the path. You can think of A* search as combining the two search methods we have already seen: Dijkstra's always expands nodes using their distance from the start state while greedy search expands states based on their heuristic score.

Using the example in Figure 3 again, A* would also first add node a to the frontier and then expand node a, adding nodes b, d and e to the frontier. However, when deciding which node to expand next, A* search would compare $f(b) = 2 + 6 = 8$, $f(d) = 4 + 2 = 6$ and $f(e) = 9 + 1 = 10$. Based on these priorities, A* would (correctly) choose to expand node d next and enqueue G. Finally, A* checks that the priority of G is the lowest amongst nodes in the frontier: $f(G) = 6 + 0$; if there was a node with lower priority still in the frontier, then there could exist a shorter path. However, in this example, $f(G)$ is lower than both $f(b)$ and $f(e)$ so A* dequeues G and the algorithm terminates, returning the optimal path: S $\rightarrow$ a $\rightarrow$ d $\rightarrow$ G.

Revising the example from Figure 1, A* search finds the same optimal path that Dijkstra's algorithm does but does so much quicker as it uses the heuristic to focus its search direction.

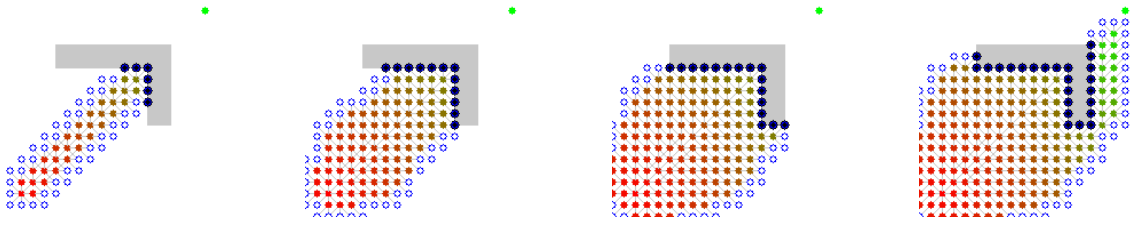

Figure 4: A* Search for finding the shortest path: unlike Dijkstra's algorithm, the frontier (the open blue circles) expands more directly towards the end state (the green dot), in an oblong, non-uniform shape.

Source: https://en.wikipedia.org/wiki/File:Astar_progress_animation.gif

3.1 Conditions for Optimality

Thus far, we have seen a few examples where A* search performs well relative to the alternatives. In fact, we can guarantee that A* finds the optimal solution if the heuristic function h satisfies certain conditions. If the graph is a tree (an acyclic graph), then A* search is optimal as long as the heuristic is **admissible**. A heuristic is admissible if it never *overestimates* the true distance between a node and the end state i.e., if $q(n)$ is the actual shortest distance between n and the end state, then h is admissible if $h(n) \leq q(n) \; \forall$ nodes n. As an example, the "as the crow flies" distance between two cities is an admissible heuristic because the true travel distance two cities is at least the distance of the most direct path.

For general graphs, A* search is optimal if the heuristic h is **consistent**. A heuristic is consistent if for every node n and all of its neighbors n', $h(n)$ is less than or equal to $h(n')$ plus the distance from n to n'; this is also known as the **triangle inequality**. Effectively, consistency requires that your estimate is "rational": if your best guess for the distance between n and the end state is some value $h(n)$, then it makes sense that this value $h(n)$ is less than (or equal to) the distance between n and n' plus your estimate for the distance between n' and the end state; if that were not the case, then your estimate $h(n)$ should be lower! Note that consistency is a *strictly stronger* condition than admissibility: every consistent heuristic is admissible but not all admissible heuristics are consistent.

Knowledge check

1. **Select all that apply:** Which of the following are reasons you should prefer A* search to Dijkstra's algorithm?
   - A A* search is guaranteed to find an optimal solution if one exists and Dijkstra's algorithm is not.
   - B A* search is more computationally efficient than Dijkstra's algorithm.
   - C A* search requires a heuristic while Dijkstra's algorithm does not.
   - D A* search can work on unweighted graphs while Dijkstra's algorithm only works on weighted graphs.
   - E None of the above
   - **Answer**: B; A is incorrect as Dijkstra's algorithm is guaranteed to find an optimal solution if one exists and A* search can fail to find an optimal solution if the heuristic function is not admissible/consistent. C is incorrect: the statement is true but is a reason to prefer Dijkstra's algorithm over A* and not vice-versa. D is incorrect because both algorithms can work on unweighted graphs.
2. **Select one:** Suppose you have a weighted graph where all of the edge weights are strictly greater than 0. Is a *constant* heuristic function, $h(n) = C \geq 0 \; \forall \; n$, consistent, admissible or neither?
   - A Consistent
   - B Admissible but not consistent
   - C Neither admissible nor consistent
   - D It depends on the specifics of the problem
   - **Answer**: C; as an easy check, if the heuristic function evaluated at the goal state is greater than 0 and all edge weights are greater than 0, then such a heuristic cannot be admissible as it is overestimating the true distance between the goal state and itself (O); note that the assumption of edge weights being greater than 0 is crucial because otherwise we could in theory find a path from the goal state to itself with negative weight. If a heuristic is not admissible, then it cannot be consistent.