A* Search

QUIZ QUESTION

Which of the following statements are correct about A* search?

- [x] A heuristic function provides the robot with knowledge about the environment, guiding it in the direction of the goal.
- [x] A* uses the sum of the path cost and the heuristic function to determine which nodes to explore next.
- [x] A* uses a priority queue as the data structure underlying the frontier.
- [ ] A* is optimal.
- [x] Choosing an appropriate heuristic function (ex. Euclidean distance, Manhattan distance, etc.) is important. The order in which nodes are explored will change from one heuristic to another.

SUBMIT

As you saw in the video above, A* search orders the frontier using a priority queue, ordered by f(n), the sum of the path cost and the heuristic function. This is very effective, as it requires the search to keep paths short, while moving towards the goal. However, as you may have discovered in the quiz - A* search is not guaranteed to be optimal. Let's look at why this is so!

A* search will find the optimal path *if* the following conditions are met,

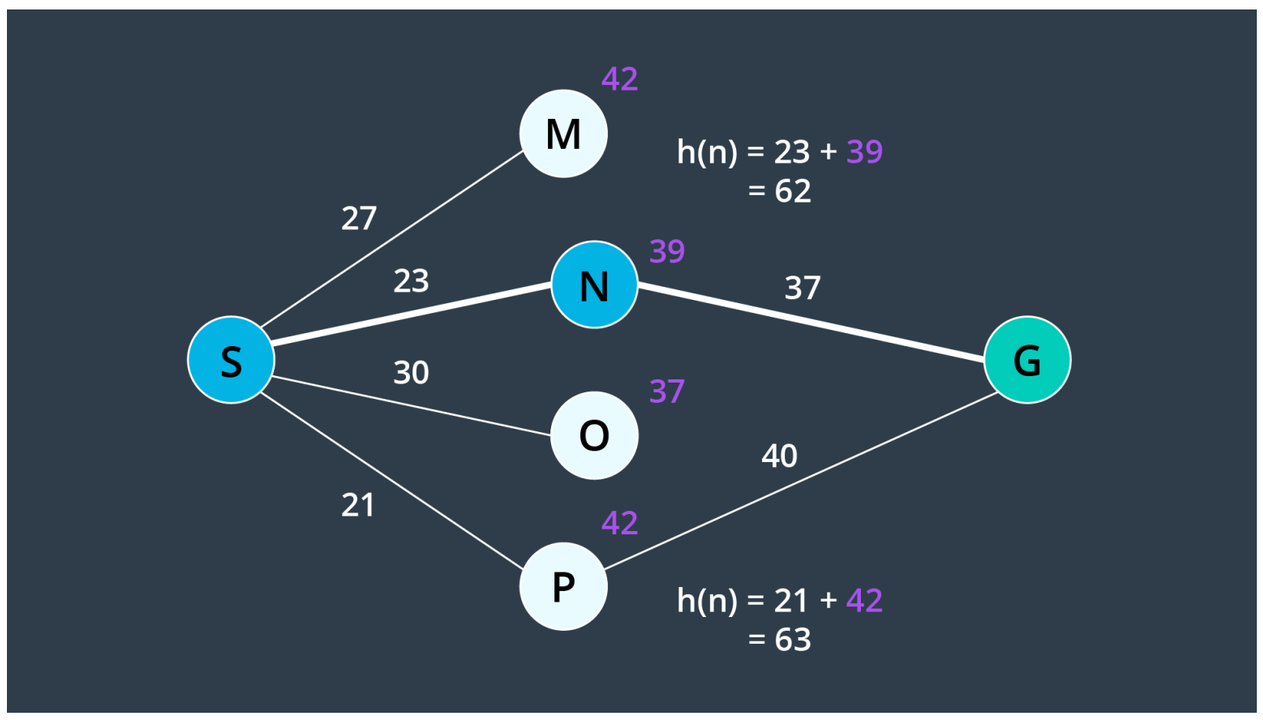
- Every edge must have a cost greater than some value, ϵ, otherwise, the search can get stuck in infinite loops and the search would not be complete.
- The heuristic function must be consistent. This means that it must obey the triangle inequality theorem. That is, for three neighbouring points (x_1, x_2, x_3), the heuristic value for x_1 to x_3 must be less than the sum of the heuristic values for x_1 to x_2 and x_2 to x_3.
- The heuristic function must be admissible. This means that h(n) must always be less than or equal to the true cost of reaching the goal from every node. In other words, h(n) must never overestimate the true path cost.

To understand where the admissibility clause comes from, take a look at the image below. Suppose you have two paths to a goal where one is optimal (the highlighted path), and one is not (the lower path). Both heuristics overestimate the path cost. From the start, you have four nodes on the frontier, but Node N would be expanded first because its h(n) is the lowest - it is equal to 62. From there, the goal node is added to the frontier - with a cost of 23 + 37 = 60. This node looks more promising than Node P, whose h(n) is equal to 63. In such a case, A* finds a path to the goal which is not optimal. If the heuristics never overestimated the true cost, this situation would not occur because Node P would look more promising than Node N and be explored first.

A* search will find the optimal path *if* the following conditions are met,

- Every edge must have a cost greater than some value, ϵ, otherwise, the search can get stuck in infinite loops and the search would not be complete.
- The heuristic function must be consistent. This means that it must obey the triangle inequality theorem. That is, for three neighbouring points (x_1, x_2, x_3), the heuristic value for x_1 to x_3 must be less than the sum of the heuristic values for x_1 to x_2 and x_2 to x_3.
- The heuristic function must be admissible. This means that h(n) must always be less than or equal to the true cost of reaching the goal from every node. In other words, h(n) must never overestimate the true path cost.

To understand where the admissibility clause comes from, take a look at the image below. Suppose you have two paths to a goal where one is optimal (the highlighted path), and one is not (the lower path). Both heuristics overestimate the path cost. From the start, you have four nodes on the frontier, but Node N would be expanded first because its h(n) is the lowest - it is equal to 62. From there, the goal node is added to the frontier - with a cost of 23 + 37 = 60. This node looks more promising than Node P, whose h(n) is equal to 63. In such a case, A* finds a path to the goal which is not optimal. If the heuristics never overestimated the true cost, this situation would not occur because Node P would look more promising than Node N and be explored first.



As you saw in the image above, admissibility is a requirement for A* to be optimal. For this reason, common heuristics include the Euclidean distance from a node to the goal (as you saw in the video), or in some applications the Manhattan distance. When comparing two different types of values - for instance, if the path cost is measured in hours, but the heuristic function is estimating distance - then you would need to determine a scaling parameter to be able to sum the two in a useful manner.

If you are interested in learning more about heuristics, visit Amit's Heuristics Guide on Stanford's website.

While A* is a much more efficient search in most situations, there will be environments where it will not outperform other search algorithms. This happens if the path to the goal happens to go in the opposite direction first.

Variants of A* search exist - some accommodate the use of A* search in dynamic environments, while others help A* become more manageable in large environments.

Additional Resources

The following visualization is a great tool that allows you to draw your own obstacles, set your own rules, and perform search using different algorithms.

Path Finding Visualization

For more information on A* variants, take a look at:
MovingAI A* Variants
Variants of A* - Stanford

Take some time to investigate the efficiency of A* over BFS in different scenarios! And if you're feeling extra adventurous, research some of the other algorithms that are provided in the simulation and compare their results to those of BFS & A*.

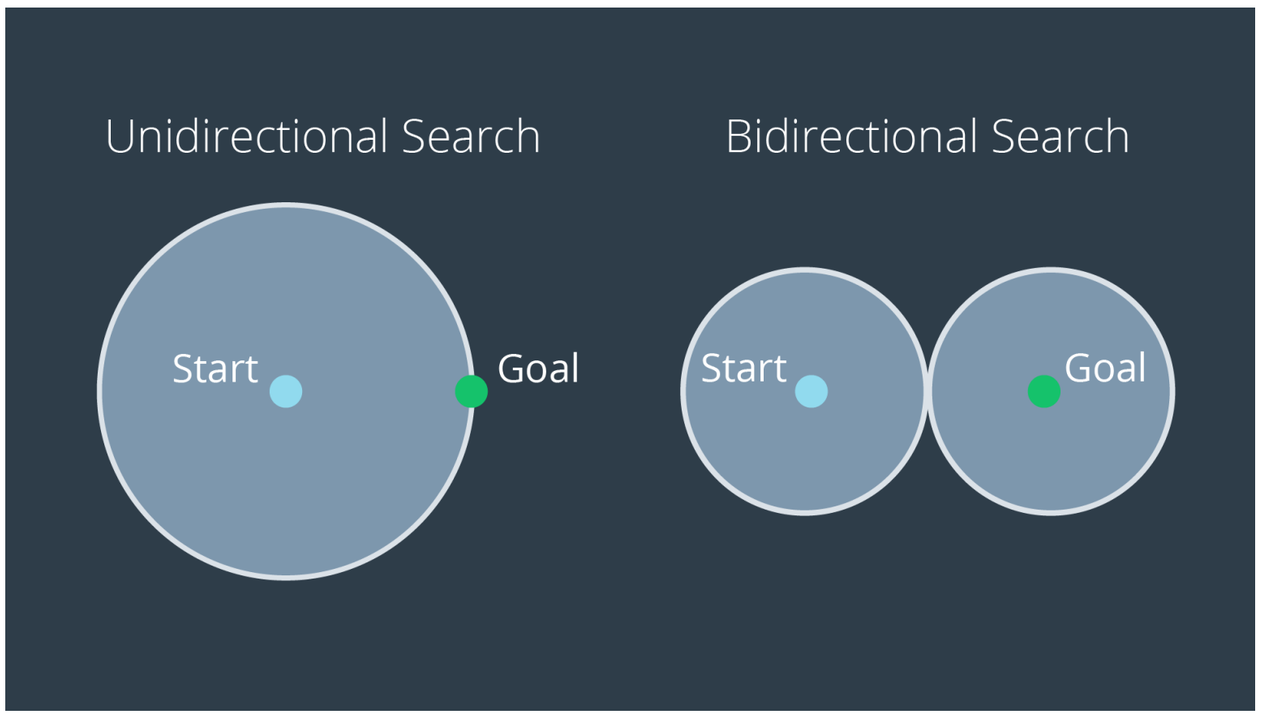
NEXT

Overall Concerns Regarding Search

Bidirectional Search

One way to improve a search's efficiency is to conduct two searches simultaneously - one rooted at the start node, and another at the goal node. Once the two searches meet, a path exists between the start node and the goal node.

The advantage with this approach is that the number of nodes that need to be expanded as part of the search is decreased. As you can see in the image below, the volume swept out by a unidirectional search is noticeably greater than the volume swept out by a bidirectional search for the same problem.



Path Proximity to Obstacles

Another concern with the search of discretized spaces includes the proximity of the final path to obstacles or other hazards. When discretizing a space with methods such as cell decomposition, empty cells are not differentiated from one another. The optimal path will often lead the robot very close to obstacles. In certain scenarios this can be quite problematic, as it will increase the chance of collisions due to the uncertainty of robot localization. The optimal path may not be the best path. To avoid this, a map can be 'smoothed' prior to applying a search to it, marking cells near obstacles with a higher cost than free cells. Then the path found by A* search may pass by obstacles with some additional clearance.

Paths Aligned to Grid

Another concern with discretized spaces is that the resultant path will follow the discrete cells. When a robot goes to execute the path in the real world, it may seem funny to see a robot zig-zag its way across a room instead of driving down the room's diagonal. In such a scenario, a path that is optimal in the discretized space may be suboptimal in the real world. Some careful path smoothing, with attention paid to the location of obstacles, can fix this problem.

NEXT