CS221 Summer 2023: Artificial Intelligence: Principles and Techniques

Homework 3: Route Planning

SUNet ID:	06781043
Name:	Zhecheng Liao
Collaborators:	[list all the people you worked with]

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

In route planning, the objective is to find the best way to get from point A to point B (think Google Maps). In this homework, we will build on top of the classic shortest path problem to allow for more powerful queries. For example, not only will you be able to explicitly ask for the shortest path from the Gates Building to Coupa Cafe at Green Library, but you can ask for the shortest path from Gates back to your dorm, stopping by the package center, gym, and the dining hall (in any order!) along the way.

Before you get started, please read the Assignments section on the course website thoroughly.

Problem 0: Grid City

Consider an infinite city consisting of locations (x, y) where x, y are integers. From each location (x, y), one can go east, west, north, or south. You start at $(0, 0)$ and want to go to (m, n). We can define the following search problem to capture this:

1. $s_{\text{start}} = (0, 0)$
2. $\text{Actions}(s) = \{(+1, 0), (-1, 0), (0, +1), (0, -1)\}$
3. $\text{Succ}(s, a) = s + a$
4. $\text{Cost}((x, y), a) = 1 + \max(x, 0)$ (it is more expensive as x increases)
5. $\text{IsEnd}(s) = \mathbf{1}[s = (m, n)]$

1. What is the minimum cost path? Is it unique (i.e., are there multiple paths that achieve the minimum cost)?

What we expect: 1 - 2 sentences describing the minimum cost path(s) and whether they're unique

Your Solution:

$$mincost = |n| + \frac{(1+|m|)|m|}{2}$$

Because as x increases, the cost will be more expensive, so our strategy is go up to n at first, and then go to m. And the $mincost$ is unique.

2. How will Uniform Cost Search (UCS) behave on this problem? Mark the following as true or false:
   (a) UCS will never terminate because the number of states is infinite.
   (b) UCS will return the minimum cost path and explore only locations between $(0, 0)$ and (m, n); that is, (x, y) such that $0 \leq x \leq m$ and $0 \leq y \leq n$
   (c) UCS will return the minimum cost path and explore only locations whose past costs are less than the minimum cost from $(0, 0)$ to (m, n).

What we expect: T/F for each subpart.

Your Solution: F; F; T

3. Now consider running UCS on an arbitrary graph. Mark the following as true or false:
   (a) If you add a connection between two locations, the minimum distance cannot go up.
   (b) If you make the cost of an action from some state small enough (possibly negative), that action will show up in the minimum cost path.
   (c) If you increase the cost of each action by 1, the minimum cost path does not change (even though its cost does).

What we expect: T/F for each subpart.

Your Solution: T; T; F

Problem 1: Finding Shortest Paths

We first start out with the problem of finding the shortest path from a start location (e.g., the Gates Building) to some end location. In Google Maps, you can only specify a specific end location (e.g., Coupa Cafe at Green Library).
In this problem, we want to give the user the flexibility of specifying multiple possible end locations by specifying a set of "tags" (e.g., so you can say that you want to go to any place with food versus a specific location like Tressider).

1. Implement ShortestPathProblem so that given a startLocation and endTag, the minimum cost path corresponds to the shortest path from startLocation to any location that has the endTag.

   In particular, you need to implement startState(), isEnd(state), successorsAndCosts(state).

   Recall the separation between search problem (modeling) and search algorithm (inference). You should focus on modeling (defining the ShortestPathProblem); the default search algorithm, UniformCostSearch (UCS), is implemented for you in util.py.

   What we expect: An implementation of the ShortestPathProblem class.

2. Run python mapUtil.py > readableStanfordMap.txt to write a (long-ish) file of the possible locations on the Stanford map along with their tags. Each tag is a [key]=[value]. Here are some examples of keys:

   - landmark: Hand-defined landmarks (from data/stanford-landmarks.json)
   - amenity: Various amenity types (e.g., "park", "food")
   - parking: Assorted parking options (e.g., "underground")

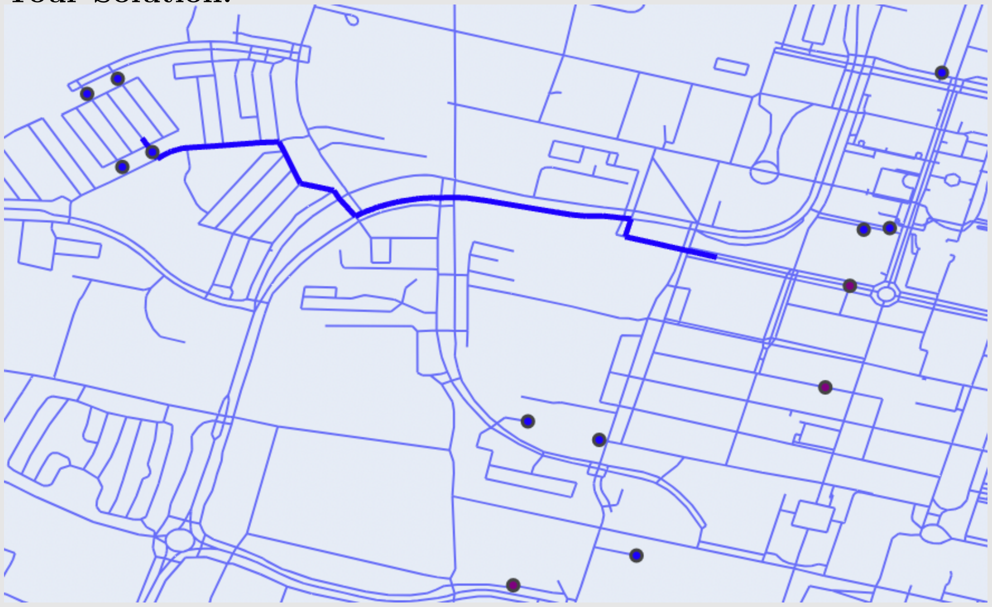
   Choose a starting location and end tag (perhaps that's relevant to your daily life) and implement getStanfordShortestPathProblem() to create a search problem. Then, run python grader.py 1b-custom to generate path.json. Once generated, run python visualization.py (with the relevant flags) to visualize it (opens in browser). Try different start locations and end tags. Pick two settings corresponding to the following:

   - A start location and end tag that produced new insight into traveling around campus. Describe whether the system was useful.
   - A start location and end tag where the minimum cost path found isn't desirable. Is this due to incorrect modeling assumptions? Explain.

   You should feel free to add new landmarks and if you are not at Stanford, follow the instructions in the README.md to use your own map and landmarks.

What we expect: A screenshot of the visualization of your two solutions as well as i) one or more sentences describing something interesting you've learned about traveling (around campus, or elsewhere) and ii) something incorrect about either the map or modeling assumptions (such a landmark being out of place, etc.).

Your Solution:



from this graph, we can identify that the minimum cost path perform well.

3. Your system now allows anyone to find the shortest path between any pair of locations on campus. By shortening their travel distance, it promises to optimize travel efficiency.

   But now, what happens when a large fraction of the population start using your system to plan their routes? In particular, what negative externalities might result from this system being widely deployed (see articles linked on website for inspiration).

   Discuss the impact of these externalities on users of your system and other people. Remember that problems often arise from the mismatch between the real world and one's model of it. How would you reduce this mismatch?

What we expect: Provide 3-5 sentences describing at least two externalities (one for users and one for other people), and what you could do to address these problems.

Your Solution: For users, as large amount of users using the same algorithm, this will lead the shortest path become crowded, negating the time-saving advantage. For those people who are not using this algorithm, this shortest path may bring crowd people to some areas need to be quite, which will interrupt people doing things there.

Problem 2: Finding Shortest Paths with Unordered Waypoints

Let's introduce an even more powerful feature: unordered waypoints! In Google Maps, you can specify an ordered sequence of waypoints that a path must go through – for example, going from point A to point X to point Y to point B, where [X, Y] are ”waypoints” (such as a gas station or a friend's house).
However, we want to consider the case where the waypoints are unordered: X, Y, so that both $A \to X \to Y \to B$ and $A \to Y \to X \to B$ are allowed. Moreover, X, Y, and B are each specified by a tag like in Problem 1 (e.g., amenity=food).
This is a neat feature if you think about your day-to-day life; you might be on your way home after a long day, but need to stop by the package center, Tressider to grab a bite of food, and the bookstore to buy some notebooks. Having the ability to get a short, quick path that hits all these stops might be really convenient (rather than searching over the various waypoint orderings yourself).

1. Implement WaypointsShortestPathProblem so that given a startLocation, set of waypointTags, and an endTag, the minimum cost path corresponds to the shortest path from startLocation to a location with the endTag, such that all of waypointTags are covered by some location in the path.

   Note that a single location can be used to satisfy multiple tags.

   Like in Problem 1, you need to implement startState(), isEnd(state), and successorsAndCosts(state).

   There are many ways to implement this search problem, so you should think carefully about how to design your State. We want to optimize for a compact state space so that search is as efficient as possible.

   What we expect: An implementation of the WaypointsShortestPathProblem class. To get full credit, your implementation must have the right asymptotic dependence on the number of locations and waypoints. Note that your code will timeout if you do this incorrectly.

2. If there are n locations and k waypoint tags, what is the maximum number of states that UCS could visit?

   What we expect: A mathematical expression that depends on n and k, with a brief explanation justifying it.

Your Solution:

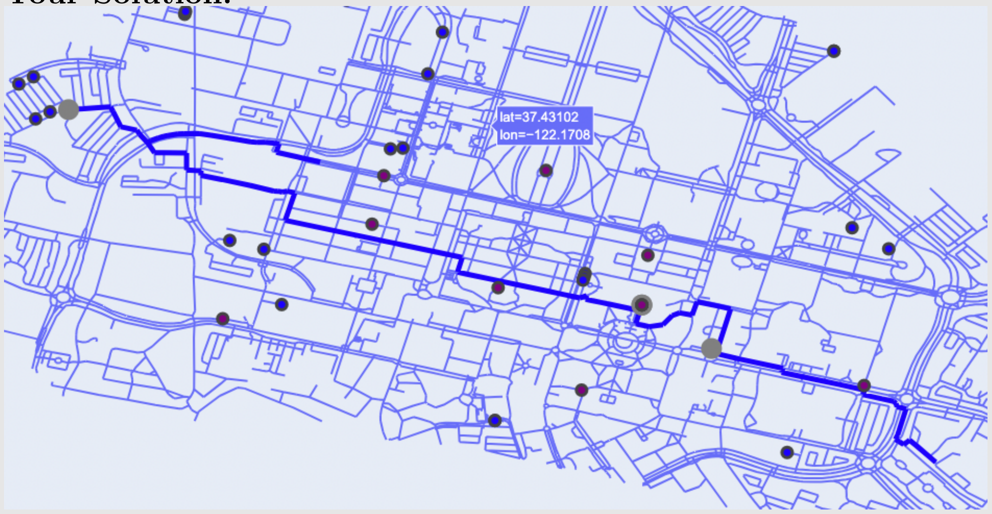
$$maximum_state = 2^{n-k} k!$$

There are k waypoints, to visit all the waypoints, the number of UCS should visit is the permutations of the waypoints, which is $k!$; and there are n locations, and two state(visited, not visited) for UCS, so for the remaining locations, the number of UCS should visit is 2^{n-k}.

3. Choose a starting location, set of waypoint tags, and an end tag (perhaps that captures an interesting route planning problem relevant to you), and implement getStanfordWaypointsShortestPathProblem() to create a search problem. Then, similar to Problem 1b, run python grader.py 2c-custom to generate path.json. Once generated, run python visualization.py to visualize it (opens in browser).

What we expect: A screenshot of the visualized path with a list of the waypoint tags you selected. In addition, provide one or more sentences describing unexpected or interesting features of the selected path. Does it match your expectations? What does this path fail to capture that might be important?

Your Solution:



I find a interesting part is that it will choose to go to a waypoint, and then back to a place, and go to another waypoint, and finally go to the terminate. I don't think that is the shortest way to go through all these waypoints.

4. Ride sharing companies use route finding systems similar to the one you built in this problem to route drivers. Such companies have been criticized for using behavioral nudges in their driver-facing applications; for example, a New York Times article (see assignment page) reported how the app uses features such as "forward dispatch" (a feature that dispatches a new ride to a driver before the current one ends) to constantly keep drivers busy. This makes it more difficult for drivers to take breaks.

How would you implement better labor practices for drivers, balancing company goals with the protection of driver interests? How could your new unordered waypoints feature help? What waypoints would you include from one drop-off location to the next pick-up location for drivers, and what information about drivers would you need to determine appropriate waypoints?

Note that the unordered waypoints feature is an example of a dual use technology: what are some waypoints that a company might use to increase its profits at the expense of driver health?

What we expect: Provide 3-5 sentences describing waypoints to include, at least one dimension of drivers' identity that could inform selection of waypoints, and a potential use of the unordered waypoints feature that would not be beneficial for drivers.

Your Solution: First, I will add some break time for drivers, it is good for their physical and mental health. Waypoints can include some position, such as restrooms, food options. To determine appropriate waypoints, information about driving times, driver preferences, and fatigue levels could be considered. Also, waypoints could incorporate driver feedback to continually improve the algorithm.

Problem 3: Speeding up Search with A*

In this problem, we will explore how to speed up search by reducing the number of states that need to be expanded using various A* heuristics.

1. In this problem, you will implement A* to speed up the search. Recall that in class, A* is just UCS with a different cost function (a new search problem), so we will implement A* via a reduction to UCS.

   In particular, you should implement aStarReduction() which takes a search problem and a heuristic as input and returns a new search problem.

   What we expect: An implementation of the NewSearchProblem class in the aStarReduction(problem, heuristic) function. As in prior problems, you need to implement startState(), isEnd(state), and successorsAndCosts(state).

2. We are now ready to speed up search by simply implementing various heuristics. First, implement StraightLineHeuristic for Problem 1, which returns the straight line distance from a state to any location with the end tag. Note: you might want to precompute some things so that evaluating the heuristic is fast.

   What we expect: An implementation of the StraightLineHeuristic class.

3. Next, implement NoWaypointsHeuristic for Problem 2, which returns the minimum cost path from a state to any location with the end tag but ignoring all the waypoints (essentially the solution to Problem 1, so you can reuse that if you'd like). Note: you might want to precompute some things so that evaluating the heuristic is fast.

   What we expect: An implementation of the NoWaypointsHeuristic class.

4. Recall that heuristics are most effective when they are equal to the future cost. Consider a $n \times n$ grid map (createGridMap) where the start and end are at the opposite corners; for $n = 10$, we would have startLocation = "0,0", endTag = "label=9,9".

   Provide a concrete example of n waypointTags so that running A* with the NoWaypointsHeuristic results in the same running time complexity as solving the relaxed shortest path problem.

   What we expect: An example of n waypointTags that satisfies the constraints of the question.

Your Solution: When the waypoints positioned on the route of the shortest path, the nowaypointHeuristic can perform same complexity as solving relaxed shortest path problem.

For example, $n = 5$,startLocation=”0,0”, endTag=”4,4”, the waypoints are [”1,0”, ”1,1”, ”2,1”, ”2,2”, ”3,2”, ”3,3”, ”4,3”]. In this case, the nowaypointsHeuristic will perform same running time complexity as solving the relaxed shortest path problem.

Submission

Submission is done on Gradescope.

Written: When submitting the written parts, make sure to select **all** the pages that contain part of your answer for that problem, or else you will not get credit. To double check after submission, you can click on each problem link on the right side and it should show the pages that are selected for that problem.

Programming: After you submit, the autograder will take a few minutes to run. Check back after it runs to make sure that your submission succeeded. If your autograder crashes, you will receive a 0 on the programming part of the assignment. Note: the only file to be submitted to Gradescope is `submission.py`.

More details can be found in the Submission section on the course website.