CS221 Spring 2025: Artificial Intelligence: Principles and Techniques

Homework 3: Route Planning

April 14, 2025

SUNet ID: [your SUNet ID]
Name: [your first and last name]
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 the Green Library Coupa Cafe, 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 1: 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. These movements are what we refer to as 'actions'. Specifically, an action is a change in the (x, y) coordinates: (+1, 0) represents moving east, (-1, 0) represents moving west, (0, +1) represents moving north, and (0, -1) represents moving south.

You start at $(0, 0)$ and want to go to (m, n), where $m, n \geq 0$. 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. $\text{Cost}((x, y), a) = 1 + \max(x, 0)$ (i.e., it is more expensive as x increases)
5. $\text{IsEnd}(s) = \mathbf{1}[s = (m, n)]$

(a) What is the minimum cost of reaching location (m, n) (note that $m, n \geq 0$) starting from location $(0, 0)$ in the above city? Describe one possible path achieving the minimum cost. Is it unique (i.e., are there multiple paths that achieve the minimum cost)? Note that the cost of the actions depends on the x-coordinate of the current state.

What we expect: Provide an expression for the minimum cost. In addition, please provide 1 - 2 sentences describing one possible minimum cost path and state whether it is unique.

Your Solution:

(b) How will Uniform Cost Search (UCS) behave on this problem? Mark the following as true or false:

1. UCS will never terminate because the number of states is infinite.
2. 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 $0 \leq y \leq n$
3. UCS will return the minimum cost path and explore only locations whose past costs are less than or equal to the minimum cost from $(0, 0)$ to (m, n).

What we expect: T/F for each subpart.

Your Solution:

(c) Now consider running UCS on an arbitrary graph to find the minimum cost path between two nodes in the graph. In particular, the graph is not necessarily the one which was studied in the previous two parts of this question. Below, we will use the terms "nodes" and "states" interchangeably. Mark the following as true or false:

1. If you add a connection between two nodes of the graph, the minimum cost to move between any start node and end node cannot go up.
2. If you make the cost of an action from some state small enough (possibly negative), that action will show up in the path returned by UCS, assuming that a path from the start state to the end state exists.
3. 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:

Problem 2: 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 Tresidder).

(a) 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), actionSuccessorsAndCosts(state). For Problems 2-4, our action space is the set of all named locations, where a named location represents a transition from the current location to that new location. Note: please read the methods in util.py (aside from PriorityQueue) for details, as well as the details of the CityMap class in mapUtil.py.

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.

(b) 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., "parking_entrance", "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 2b-custom to generate path.json. Once generated, run python visualization.py 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 (e.g. missing tags/paths, inaccurate locations/distances)? Explain.

You should feel free to add new landmarks. If you are not familiar with the Stanford campus, please feel free to use the Campus Map website to learn more about buildings, amenities, and paths around campus. The function locationFromTag() in mapUtil.py may be helpful.

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:

(c) 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, there might be issues 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? Please refer to these brief articles for inspiration: "Why Traffic Apps Make Congestion Worse," "The Perfect Selfishness of Mapping Apps", as well as the module on Externalities (video and pdf).

Discuss the impact of these externalities on (1) users of your system and (2) non-users. Remember that these sorts of problems arise from the mismatch between the real world and one's model of it.

What are potential ways you could reduce this mismatch?

What we expect: Provide 3-5 sentences describing (a) two externalities (one for users of the route-planning system and one for other people/non-users of the system), as well as (b) at least one potential solution to reduce the impact of these problems.

Your Solution:

Problem 3: 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 2 (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, Tresidder 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).

(a) 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 (potentially including the startLocation). You can assume that waypoint tags are distinct.

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

Like in Problem 2, you need to implement startState(), isEnd(state), and actionSuccessorsAndCosts(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.

(b) If there are n locations and k waypoint tags, what is the maximum possible number of states given a suitable state defintion for part a that UCS could visit?

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

Your Solution:

(c) 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 2b, run python grader.py 3c-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. Try to include at least two waypoints in your path. 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:

(d) Ride sharing companies use route-finding systems similar to the one you built in this problem to route their drivers. However, these systems have been criticized for using exploitative behavioral nudges. For example, a New York Times article reported how one app uses "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.

We want you to discuss how these companies could implement better labor practices, finding a way to use the unordering waypoints feature to balance company goals with the protection of driver interests. Specifically, we want you to think about ways your new unordered waypoints feature could help drivers in their working experience.

What waypoints could you include from one drop-off location to the next pick-up location to improve the driver experience? What information about drivers would you need to determine appropriate waypoints?

Note, however, that the unordered waypoints feature is an example of a dual use technology. For more, refer to the module on dual use technologies to help answer this question (video and pdf).

Thus, we also want you to answer: What are some ways that a company could use this waypoints feature to increase profits at the expense of driver health? What are any downsides in collecting the appropriate information you need to recommend the waypoints?

What we expect: Provide 3-5 sentences describing (a) the waypoints you would include to improve the driver experience, (b) at least one dimension of drivers' identity that could inform selection of waypoints, and (c) a potential negative use of the waypoints feature that a company could use to increase profits while harming the driver experience or driver health, or downsides in collecting the appropriate information needed to recommend the waypoints.

Your Solution:

Problem 4: 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.

(a) 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 actionSuccessorsAndCosts(state).

(b) We are now ready to speed up search by simply implementing various heuristics. First, implement StraightLineHeuristic for part (b), 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.

(c) Next, implement NoWaypointsHeuristic for part (c), 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 2 (a), so you can reuse that if you'd like). Note: you might want to precompute some things so that evaluating the heuristic is fast. Helpful comments are provided in the code.

> **What we expect:** An implementation of the NoWaypointsHeuristic class.

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.