Frontier Navigation

Welcome to the Frontier Navigation family! We're glad to have someone with AI experience on board. As you know, our landmark product, the Skamania Hiker's GPS, has been a commercial success, and our customer base is waiting excitedly for the Skamania II, which will be cheaper and more powerful than the original model. One innovation that will set us apart from the competition is our off-trail pathfinding capabilities which can find the most hiker-friendly path between two points regardless of the terrain. Unfortunately, our pathfinding algorithms have either been too slow or returned suboptimal paths. With the upcoming launch of the Skamania II drawing near, we could use your insights on this feature. We hope you can integrate your pathfinding algorithm into our existing code and present your findings by February 2nd.

We are looking forward to seeing your work.

—----------
Sincerely,

Michael Livanos
VP of Operations
Frontier Navigation Incorporated

Project_requirements.pdf

Skamania II - Off-trail Pathfinding Requirements

The World

Our device renders maps using the Map class, a class encapsulating a two-dimensional world layered on top of a rectangular grid. Each point in the world has a height, **represented by an integer value** between 0 and 255. You can move to any of the eight squares adjacent to your location (e.g. the four cardinal directions and the diagonals). As you would expect, the cost to traverse between tiles is dependent on the differences in height between the tiles.

Cost Functions

The Skamania II GPS's off-trail pathfinding feature needs to find the most hiker-friendly trail for an arbitrary terrain. The terrain is represented as a 3D surface discretized into uniformly spaced tiles. The hiker can move with chessboard motion across the XY plane of this surface with the "cost" per step being defined by the change in height for each step using one of two cost functions:

$cost(h_0, h_1) = e^{h_1-h_0} = e^{\Delta h}$ (exp) - short for "exponential"
$cost(h_0, h_1) = 1 + (h1 - h0)^2$ (sqdf) - short for "squared difference"

Where h_0 is the height of the current title and h_1 is the height of the tile to be moved to. The total path cost is defined as the sum of all the individual step costs.

We are hoping to implement the A* algorithm for this task, but our team lacks the expertise to code this and design admissible heuristics to ensure that the optimal path is found. As such, the project has the following components:

Designing Admissible Heuristics and Implementing A* (20 Points)

Part 1 (Section 2.1 in the code):

Design two admissible heuristics, one for each cost function.

Describe your heuristics in detail and **prove/show** why they are admissible. You can provide a mathematical proof if you would like, or you could simply enumerate all possible cases and explain why your heuristics will never overestimate the true cost in any of the cases.

Part 2 (Section 2.2 in the code):

Implement A* using the heuristics you designed above. We have provided you with an implementation of Dijkstra's Algorithm as a starting point for your A* implementation.

You will create two A* implementations:

AStarSqdf - which implements the heuristic you designed for the sqdf cost function
AStarExp - which implements the heuristic you designed for the exp cost function

For both AStarExp and AStarSqdf, we calculate the mean decrease in number of points explored when compared to the number of points that Dijkstra's Algorithm explores on the same maps. This calculation is shown in sections 2.3.1 and 2.3.2 of the code and reported as "mean_explored points_decrease".

Your goal in designing heuristics is to maximize this number while still obtaining the optimal cost.

AStarExp must obtain a mean_explored_points_decrease value of 50 or greater (meaning it explores at least 50% fewer points than Dijsktra's, on average.

AStarSqdf must obtain a mean_explored_points_decrease value of 10 or greater (meaning it explores at least 10% fewer points than Dijsktra's, on average.

Part 3 (Section 2.3 in the code):

We have prepared a series of 20 randomly generated 300x300 test maps (10 for AStarExp and 10 for AStarSqdf) to ensure that your code is working properly.

Please run the provided tests and ensure that they pass. If you submit code where any of these 20 visible tests fail, you will receive no credit for this section.

Note that while failing to meet these costs would demonstrate a problem with your code, meeting these costs does not necessarily mean that your heuristics are admissible - perhaps there is an issue with your heuristic that did not arise in testing. If you are unsure whether your heuristics are admissible, we would suggest running additional tests on edge cases (for comparison - the cost that Dijkstra's Algorithm returns for a map is the optimal cost)

We will run your A* implementations on additional hidden test maps to ensure that your heuristics are admissible.

Grading:

You will receive all 20 points if:

- You provide a reasonable explanation of your heuristics and show that they are admissible (there are no strict requirements on how you prove/show admissibility)
- You implement A* correctly
- Your two A* implementations pass the 20 visible tests and all hidden tests
- AStarExp achieves a mean_explored_points_decrease value of 50 or greater on visible and hidden tests
- AStarSqdf achieves a mean_explored_points_decrease value of 10 or greater on visible and hidden tests

Admissible Heuristics Bonus Points (5 Points)

We will calculate the mean_explored_points_decrease figure on our hidden tests and rank all students' performance. Your score will be the average of your AStarExp mean_explore_points_decrease and AStarSqdf mean_explore_points_decrease results. The top 50% of students will receive bonus points linearly proportional to their rank. There is no penalty for not being in the top 50%, this section is just to reward students who want to go above and beyond.

If your A* implementations do not pass all test cases (including hidden tests) you will not be eligible for bonus points.

Descending Mt. St. Helens - A* Variants (10 Points)

As part of our coming demonstration, we will have a hiker use the Skamania II to descend Mt. St. Helens, located here in Skamania County using the "exp" cost function. It is crucial that during this demonstration, hikers can get the optimal route quickly, so we want you to modify your A-Star approach from above to use an A-Star variant of your choosing.

All testing and evaluations in this section take place on the Mt. St. Helens map (msh.npy) **and use the 'exp' cost function**.

Part 1: (Section 3.1 in the code):

The goal of this section is to implement some variant of A* (AStarExpImproved) that is faster than vanilla A* (AStarExp). An example of a variant would be implementing bidirectional search A*; more ideas can be found here the following resource.

You will be improving your AStarExp implementation and should use it as the starting point for AStarExpImproved. You are not allowed to change the heuristic from the one you used in AStarExp; meaning that, in the code you submit, AStarExp and AStarExpImproved must use the same heuristic.

AStarExpImproved must improve runtime by at least 10% on average when compared to AStarExp. This calculation has been implemented in section 3.2 of the code and is reported as "mean_runtime_improvement". We calculate this figure on both visible and hidden tests, but if your algorithm is scoring well above 10% runtime improvement on the 5 visible test cases, you likely won't have any issues scoring above 10% on the hidden tests.

Importantly: Your A* implementation no longer needs to find the optimal path. As discussed in previous lectures, A* has an exponential runtime complexity, and thus real-world applications of A* often settle for finding near-optimal paths in order to decrease runtime.

Your implementations must find a path that has a cost that is within 10% of the optimal cost. We have also included this calculation in the code (section 3.2), reported as "mean_cost_increase"; again, for grading purposes, this figure will be calculated on both hidden tests and the 5 visible test cases.

Your goal is to maximize "mean_runtime_improvement" while keeping "mean_cost_increase" under 10%.

We include 5 test cases for you to benchmark your AStarExpImproved algorithm on. If AStarExpImproved does not achieve a "mean_runtime_improvement" of at least 10% and a "mean_cost_increase" of less than 10% on these 5 included tests, you will receive 0 points on this section.

Part 2: (Section 3.3):

Please explain what your A* variant is and provide some discussion of the costs and benefits of using it over vanilla A*.

Grading:

You will receive all 10 points if:

- AStarExpImproved obtains a mean_runtime_improvement value greater than 10 on the visible and hidden tests and has the same heuristic as AStarExp
- AStarExpImproved has a mean_cost_increase of less than 10 on the visible and hidden tests

- You provide a reasonable explanation of your A* variant and discuss trade-offs compared to vanilla A*

A* Variants Bonus Points (5 Points)

Students will be ranked by their “mean_runtime_improvement” on hidden tests (which all take place on the Mt. St. Helens map).

The top 50% of students will receive bonus points linearly proportional to their rank. There is no penalty for not being in the top 50%, this section is just to reward students who want to go above and beyond.

If AStarExpImproved does not pass all test cases (including hidden tests) you will not be eligible for bonus points.

Submission Instructions

Please submit a single .ipynb file with the naming convention:

FirstnameLastname_ProgrammingAssignment1.ipynb

Example: ZacheryFogg_ProgrammingAssignment1.ipynb

Due Date and Late Policy

This assignment is due February 2nd at 11:59pm. Late submissions are allowed up until February 12th. There is a 10% penalty for each partial day late. Late submissions are not eligible for bonus points.