Project Template

Anna Sophie Hilgard, Nick Hoernle, and Nikhila Ravi

December 7, 2016

1 Introduction

The goal of our project is to create a routing system for bicycle trips that takes into account more than just distance. Current map applications like Google often route cyclists through busy streets and intersections, resulting in unnecessarily dangerous trip and sometimes accidents. The algorithm we chose to use for this problem is A* search with a variety of cost functions and heuristics incorporating distance, safety of a given route, and changes in elevation (also important for biking but not information Google maps currently considers).

Additionally, we considered the problem of finding a central meeting place for two or more cyclists starting from different positions. To solve this optimization problem, we used k-beam search and simulated annealing.

The graph search problem is directly analogous to techniques and ideas from the class: using intersections as nodes in the graph and road segments connecting intersections as paths, we are able to construct the city maps as directed graphs. From there, we can use data generated from a variety of sources to come up with approximate costs associated with paths and related heuristics at nodes.

The optimization techniques we use are also direct applications of algorithms from the optimization portion of the course.

Problem-specific adaptations were largely limited to the collection and interpretation of relevant data and the development of a cost function and relevant heuristics.

2 Background and Related Work

While our problem was fairly straightforward, we did do some research to get ideas for potential cost attributes and data sources. Our initial formulation of the problem, for example, did not include elevation differencing, but reading previous studies led us to believe that this was an important criterion. [1].

3 Problem Specification

The problem that we are solving is of the general class of graph search problems. Given a strongly connected graph, we seek to minimize some cost function $cost(\sum_{i=0}^{n} path_i)$
given that $path_0 \in Connections_{startingnode}$

$path_i, path_{i+1} \in Connections_{node_j}$ for some j,
$path_i, path_{i-1} \in Connections_{node_k}$ for some k, and
$path_n \in Connections_{targetnode}$

Specifically for our problem, the cost function is:

TODO: get exact cost function $cost(path_i) = \alpha_i * length_i + ?? + \beta_i * abs(\Delta elevation_i)$

In the optimization portion of our project, we seek to minimize the total cost to all parties: $min(\sum_j^n cost_j)$ where $cost_j$ is the cost to participant j.

4 Approach

Data Structures: Our primary data structures are an intersection graph, which is stored as a Python dictionary, and a connection dictionary, which is also stored as a Python dictionary. The intersection graph maps nodes (intersections) by id to a list of paths (road segments) from that node. The connection dictionary maps a connection (road segment) to its source node, sink node, and various cost parameters (in our case distance, number of bicycle crashes on that road segment, and change in elevation over that road segment).

Algorithm 1 A-Star Search

```
function A-STAR-SEARCH(graph, startnode, targetnode)
    node ←a node with STATE = startnode
    PATH-COST ← heuristic(startnode, targetnode)
    frontier ← a priority queue ordered by PATH-COST + heuristic(node, targetnode) with node
as the only element
    explored ← an empty set
    loop
        if EMPTY?(frontier) then
            return failure
        end if
        node ← POP(frontier) /*chooses the lowest cost+heuristic node in frontier */
        if node == targetnode then
            return SOLUTION(node)
        end if
        add node.STATE to explored
        for each path in PATHS(node) do
            child ← child-node(node, path)
            if child.STATE is not in explored or frontier then
                frontier ← insert(child, frontier)
            else if child.STATE is in frontier with higher PATH-COST + heuristic then
                replace that frontier node with child
            end if
        end for
    end loop
end function
```

Algorithm 2 Simulated Annealing Meeting Spot

function SIMULATED ANNEALING MEETING SPOT($graph, startingpts, cost, heuristic$)
  if length$(startingpts) < 2$ **then**
    return error
  end if
  $current \leftarrow$ **mean**$(startingpts)$.CLOSEST-NODE
  $temperature \leftarrow e^{10}$
  $\gamma \leftarrow .5$ /*schedule to manage $temperature$ */
  while $temperature > e^{-2}$ **do**
    $temperature \leftarrow temperature * \gamma$
    $next \leftarrow$ a randomly selected *child* of *current*
    $current$.VALUE $\leftarrow \sum_{pt \in startingpts}$ **cost**$(pt, centroid)$
    $next$.VALUE $\leftarrow \sum_{pt \in startingpts}$ **cost**$(pt, next)$
    $\Delta E \leftarrow next$.VALUE $-$ $current$.VALUE
    if $\Delta E > 0$ **then**
      $current \leftarrow next$
    else
      $current \leftarrow next$ with probability $e^{\Delta E / temperature}$
    end if
  end while
end function

Algorithm 3 K-Beam Search Meeting Spot

function K-BEAM SEARCH MEETING SPOT($k, graph, startingpts, cost, heuristic$)
  if length$(startingpts) < 2$ **then**
    return error
  end if

  $\{candidatenodes\} \leftarrow node \,\forall\, node \in graph$ s.t.
$node.x \geq min(startingpts.x)$& $node.x \leq max(startingpts.x)$&
$node.y \geq min(startingpts.y)$& $node.y \leq max(startingpts.y)$

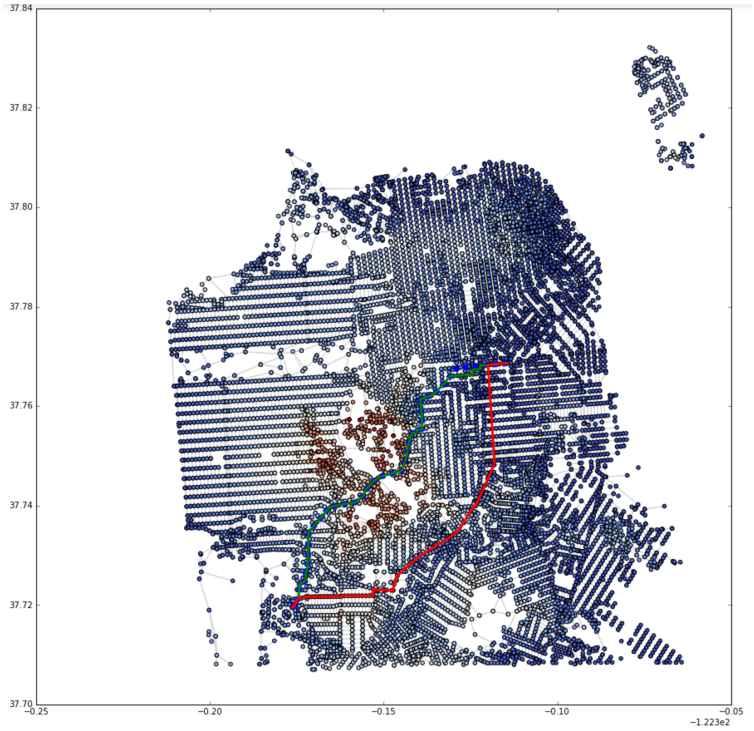
  $point_i \leftarrow$ a randomly selected $node \in \{candidatenodes\} \forall i \leq k$
  $best$.VALUE $\leftarrow min_{i \leq k}(\sum_{pt \in startingpts}$ **cost**$(pt, point_i))$
  while True do
    $\{nextcosts\} \leftarrow \sum_{pt \in startingpts}$ **cost**$(pt, child_i) \forall i \leq k, child_i \in$ PATHS$(point_i).endnode$
    $point_i \leftarrow$ i-th least $node \in \{nextcosts\} \forall i \leq k$
    $next$.VALUE $\leftarrow \sum_{pt \in startingpts}$ **cost**$(pt, point_1)$
    if $next$.VALUE $<$ $best$.VALUE **then**
      $best \leftarrow next$
    else
      break
    end if
    return *best*
  end while
end function

5 Experiments

Analysis, evaluation, and critique of the algorithm and your implementation. Include a description of the testing data you used and a discussion of examples that illustrate major features of your system. Testing is a critical part of system construction, and the scope of your testing will be an important component in our evaluation. Discuss what you learned from the implementation.

- Analysis: To test our algorithms, we built graphs on GIS data from both Cambridge, MA and San Francisco, CA (supplemented with elevation and crash data collected from other sources as necessary). We ran both the single-path and meeting point algorithms for a variety of cost functions and start and end points and were pleased to see that the paths generated by our algorithms when taking into account safety and elevation factors do indeed seem to avoid busy intersections and steep hills in a reasonable way.

Figure 1: Paths between nodes in San Francisco. Blue path optimizes for distance only, green path avoids previous bicycle accidents, and red path minimizes altitude changes. Overlaid with an altitude plot of San Francisco, we can clearly see the red path avoiding a hill.



- Evaluation: We went through a few iterations of tuning the parameters in our cost function to get the appearance of the graphs to mimic what we think we'd want as cyclists. Because the cost in our problem is qualitative rather than explicitly quantitative, we were largely forced to estimate parameters based on what we thought would be reasonable rather than deriving them from empirical data of some sort. That said, we all felt that with our final parameters the algorithm was doing a reasonable job at generating routes we would want.
- Critique: Our algorithms could have been made to run somewhat faster by incorporating pruning strategies but we were generally satisfied with run times. Additionally, it was difficult to generate a very useful admissible heuristic for bike crashes because one could almost always find an extremely convoluted path to avoid almost all of them, and so to have a heuristic which is always *le* the actual cost to the goal is often not that informative.

	Time	Total Cost
Simulated Annealing		
K-Beam Search		

Table 1: Description of the results.

5.1 Results

TODO: Comparison of K-beam search and Simulated annealing: time to find solution vs total cost of solution found for a number of trials

TODO: Comparison of routes found: For a number of routes, let's tabulate total distance traveled, total accidents passed over, and total change in elevation for the routes calculated between those nodes by our three cost functions to A*

TODO: Comparison of nodes expanded/time used for Combination Cost function with different heuristics: For a number of routes, let's tabulate total routes expanded and time used when running the combination cost function with each of our three heuristics.

6 Discussion

The graph search approach is a very straightforward and satisfactory solution to this problem. We felt that our results were very reasonable given our preexisting knowledge of Cambridge and San Francisco streets.
A couple takeaways from this project:

- As always, you're only as good as your data. We had some lofty goals for the data that we'd be able to collect and use for this project, but even the data we ended up using was harder to collect and map that we anticipated, and much of the crash data was a few years old. Cambridge and San Francisco are relatively tech-forward cities, so I can imagine this would be even more difficult in most other environments

- It was interesting to see how our algorithm scaled to the larger map of San Francisco. To plan routes through a larger area, it's clear that we would have to adapt the algorithm to get the runtime within a reasonable range. In particular, we found in our reading that many routing engines actually use inadmissible heuristics for these tasks and still find reasonable results but in a much quicker time.

In future work, we could develop faster algorithms by using pruning procedures or finding ways to run expensive operations in parallel or vectorized forms. We could develop more sophisticated models of biking comfort by also including road construction data and pothole reports.

A System Description

TODO: A clear description of how to use your system and how to generate the output you discussed in the write-up. *The teaching staff must be able to run your system.*

B Group Makeup

1. Nick Hoernle
   (a) Creation of graph dictionary structure and A*search algorithm
   (b) Simulated annealing

2. Nikhila Ravi
    (a) K-Beam Search
    (b) Visualization and analysis of results of graph search algorithms
3. Anna Sophie Hilgard
    (a) Construction of Datasets
    (b) Research and Implementation of more complicated cost functions and heuristics

References

[1] Jan Hrncir, Palov Zilecky, Qing Song, and Michal Jakob. Practical multicriteria urban bicycle routing. *IEEE Transactions on Intelligent Transportation Systems*, PP(99):1–12, 2016.