Minimal Time Paths
Project Report

Group 2:
Alex Dunkel, Roy Gullem, Keith Houpy,
Emily Ribando-Gros, and Vincent Rodomista

December 7, 2016

1 Motivation

Travel plans can be extremely complicated. As a college student, when planning a trip, you can sift through numerous flights looking for deals and also weigh the cost of driving based on gas prices and car mileage. On the other hand, a business person may do the same sort of sifting to find the fastest, most direct flights and available private transportation in order to get to their destination as fast as possible without considering the price. We are given many options when considering travel plans, and planning could end up in a series of calculations based on time, price of travel, or both.

2 Problem

We want to find the path from one point to another by minimizing time or price by either driving, flying, or a combination of both. To ensure that the price of this path is not too high, we can specify a price limit so that we find the fastest path under that price limit.

2.1 Problem Formulation

1. **Initial State**: The user's starting point
2. **States**: contact points
3. **Actions**: either fly or drive to a destination. For example, Drive(*destination*) or Fly(*destination*)
4. **Transition Model**: given a state and action moves the user to the new contact point
5. **Goal State**: the user's destination
6. **Path**: a sequence of actions from the initial state to the goal state
7. **Path Costs**: the time or price it takes to get from one contact point to another
8. **Solution**: the path from the initial state to the goal state that does not exceed the price limit

3 AI Concept : Search Tree

3.0.1 Node

A Node in the search tree represents a contact point along the path. Node attributes include

- State : Information about the contact point such as a name and latitude longitude coordinates
- Parent : The node in the search tree which generated this node
- Action : The action (fly or drive) applied to the parent to generate this node
- Path costs in time and price : the time and price needed to travel from the initial node to the current node

3.0.2 Frontier

After expanding a leaf node in our search tree, the general search tree algorithm collects the next set of available leaf nodes for expansion in the frontier. But before searching can continue, our algorithm must check whether any of the frontier nodes will result in a path cost price which exceeds the user's price limit. In other words, if the price of a path exceeds the given price limit we do not add the next node to the frontier. This way we can ensure that we never find a path with a price that is above the limit. Nodes are selected for expansion based on the search strategies described in 4.

3.1 Constraints

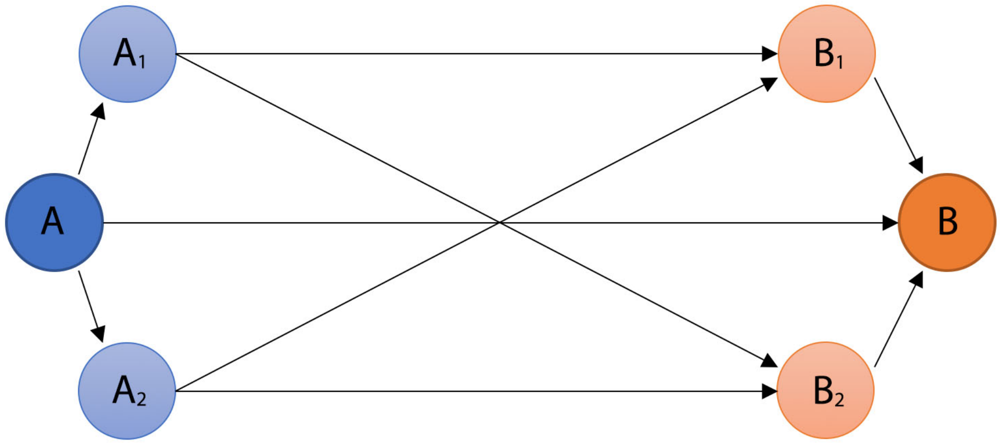
3.1.1 Branching Factor

When searching for a path, given the number of airports and cities in the US, our branching factor could be very large. Because of this, some constraints must be considered. Let A be the initial starting point and let B be the destination point. We limit the number of airports around A, say $\{A_i\}_{i=1}^{n}$, and the number of airports around the destination point, say $\{B_j\}_{j=1}^{m}$. The agent can then drive from A to one of the airports in $\{A_i\}_{i=1}^{n}$ or drive directly to B. If the agent is at one of the airports in $\{A_i\}_{i=1}^{n}$, the only option for the agent is to fly to one of the airports in $\{B_j\}_{j=1}^{m}$. If the agent is at an airport around B, then the only possible action is to drive to the destination B. Figure 1 shows an example with a graph with $n = 2$ and $m = 2$.

3.1.2 Driving Assumptions

In order to calculate driving time and cost, a few assumptions were hard coded into our application. To give a more accurate description of the time it takes to drive across the country, every 12 hours we will assume that our user will spend \$75 on a hotel. We will add this price to our path price when searching for a solution. Similarly, when calculating time we will only assume that the user will drive 12 hours a day. In other words, for every 12 hours of driving, a cost of \$75 is added to the path price and 12 hours are added to the path time. These assumptions can be seen in 3b.

Figure 1: Graph example with $n = 2$ and $m = 2$.



4 AI Concept : Search Strategies

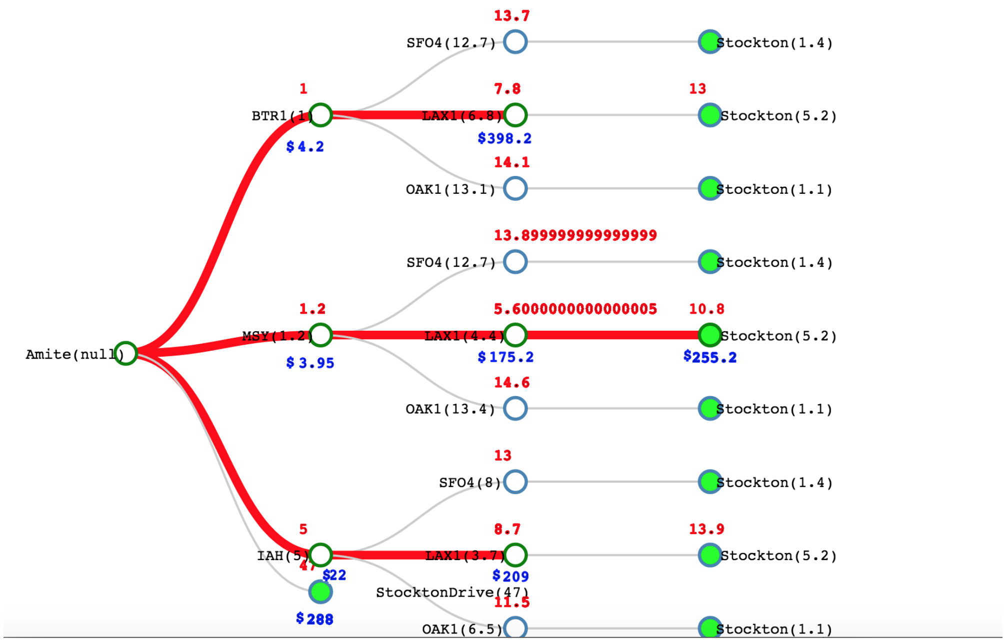
4.1 Uniform Cost Search

At the base line, our algorithm performs a tree search seeking to minimize the time it takes to travel given a price limit. Since different users may value their time more than their money or may not like driving, we considered the following features.

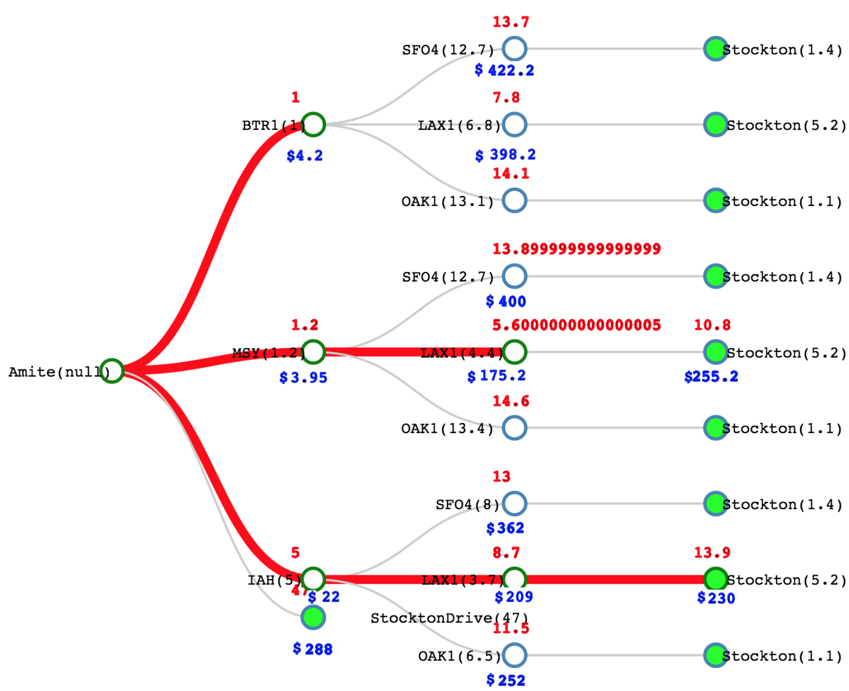
4.2 Uniform Cost Search Experiment

The following two searches were done without a price limit and with a price limit of $250. At the very bottom of the tree, there is one node to represent the driving distance, which ends up being more expensive then either of the flights we found.

$$\text{Solution : Amite} \rightarrow \text{MSY(1.2 hr)} \rightarrow \text{LAX1(4.4 hr)} \rightarrow \text{Stockton(5.2 hr)}$$
$$\text{Total Price: \$255}$$
$$\text{Total Time: 10 hrs and 48 min}$$



Solution : Amite → IAH(5 hr) → LAX1(3.7 hr) → Stockton(5.2 hr)
Total Price: $230
Total Time: 13 hrs and 54 min



FIXME:

4.2.1 Weight on Driving

4.3 A* Search with Straight Line Heuristic

In addition to the above features, we thought that we could improve upon our algorithm by adding a heuristic component to our cost function. Across edges, we are mixing different modes of transportation which have different speeds. Additionally, costs for flying depends only on plane tickets where as costs for driving depends directly on the distance and miles per gallon of a vehicle. For this reason, we can not simply just use the distance as a heuristic but we need to first convert it to the proper units based on what we would like to minimize.

FIXME:

4.3.1 Estimating Price based on Distance

Prob won't get to...

4.3.2 Estimating Time based on Distance

Since commercial flights do not reach a speed faster than 600 mph, we tested adding the distance between two places divided by this speed. We found distance using the Haversine formula. If the latitude and longitude of a node n and the destination are (ϕ_1, ϕ_2) and (λ_1, λ_2) and r is the radius of the Earth then,

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_2)\cos(\phi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

We then let

$$h(n) = \frac{d}{600} \text{ hours}$$

FIXME: Alex - Commercial flights do not fly faster than 600 mph. So we could add (straight line distance)/600 to the path cost for time. This would favor contact points that are closer to the goal point.

5 Experimentation : Amite to Stockton

To experiment with our algorithm we have chosen the example trip from Amite, LA to Stockton, CA. To see that our ai responds to minimizing different factors we compared the resulting solution sequences.

6 Software System Design and Implementation

We chose to use Java for our project in the IDE IntelliJ. We developed our GUI in JavaFX using the JFoenix design library. To compute the time and price of each edge we use many different Google APIs.

6.1 Search Classes

Given the latitude and longitude of the initial point (origin) and destination (goal), our search algorithm uses a few different classes to arrive at a solution. In 2 you can see the relevant fields and methods for each class.

1. Problem.java : stores information about the travel plans of the user
2. Node.java : represents a node in the search tree
3. Search.java : search algorithm logic
4. FindAirports.java : finds the n airports around a place

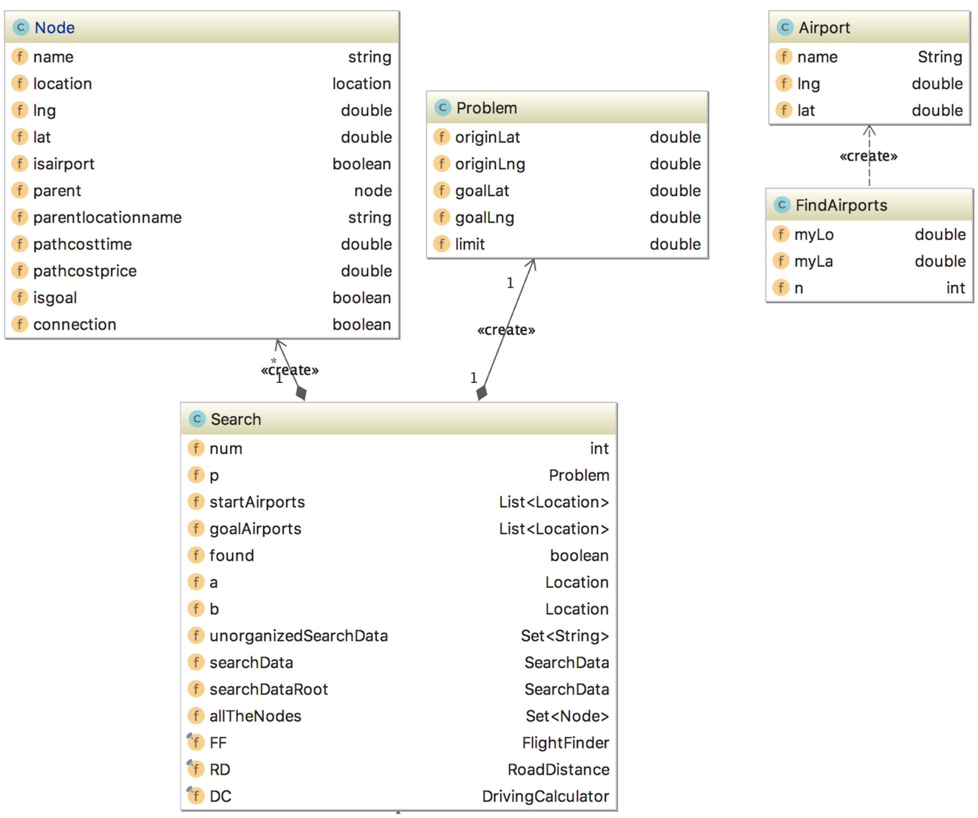
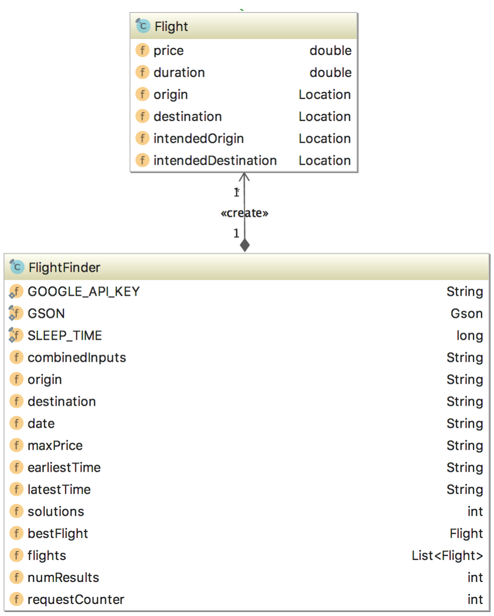
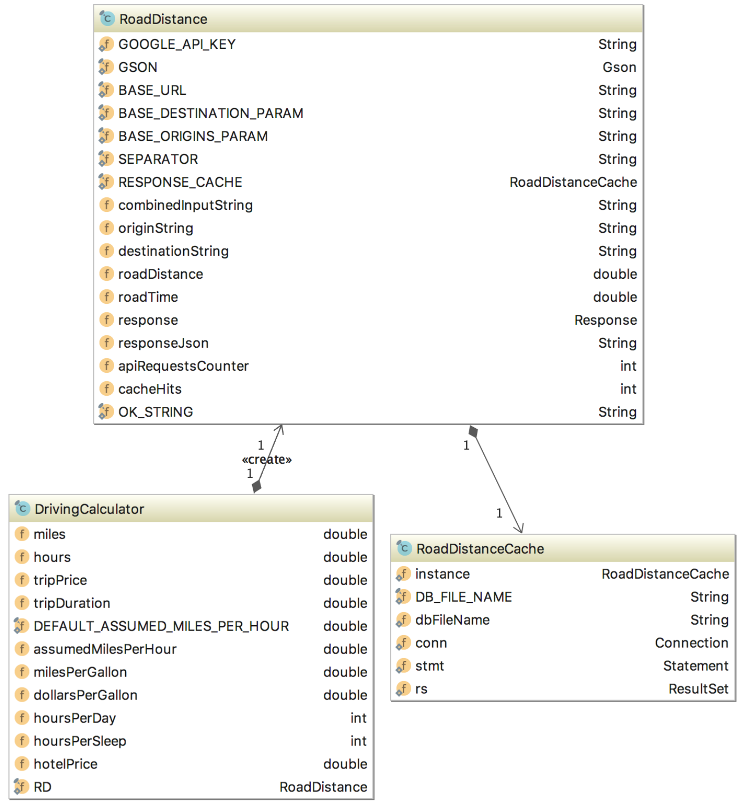


Figure 2: Search Classes



(a) Flight Finder



(b) Driving Utilities

Figure 3: Map Utilities

6.2 Map Utilities

6.2.1 Driving Calculator and Road Distance

6.2.2 Flight Finder

6.3 APIs

FIXME:Explain what each do because Chet seemed interested.

Using the latitude and longitude coordinates of places and

- QPX Express Airfare API
- Google Maps Distance Matrix API
- Google Maps Geocoding API
- Google Maps Geolocation API
- Google Maps Directions API
- Google Places API Web Service

6.4 Caching

FIXME: Even if we don't get this working we should still explain what we have with sqlite and what we wanted to do with bluemix and maybe even why we had trouble with it. A screenshot like the one from the pwrpt would be nice

Some of the APIs are quite slow, especially the QPX Express Airfare API to find flights. So to speed up our search time, we tried caching the data we found for future searches. We use two separate caches to make searches faster. One cache is used for road travel and one for flight. When we call the function used to determine the best route, it first searches the cache to see if the query has already been made in the past. If it has, it returns the result of the query. If this data is not stored, the API is called and the resulting distance is stored in the cache. This speeds up searching significantly over time because the cache will be able to be referenced more, avoiding costly calls to the API. The API calls take significantly longer than the cache calls, so it becomes faster and faster over time.

7 Future Work

This application is restricted to traveling within the US. Other countries could be included to give the user the option to travel internationally. The only modes of transportation that are used are driving and flying. To give the user a more flexibility when traveling, other modes of transportation could be incorporated into this application such as walking, bicycling, taking the train or metro, and taking a ferry.

Additionally, after caching of a large amount of data our application could make certain predictions which improve our search algorithm. For example, instead of choosing the n nearest airports to add to our search tree, Watson Analytics predict feature could also choose an airport which may be slightly farther away but which may frequently have deals on flights.