Minimal Cost Paths
Project Report

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

December 6, 2016

FIXME:Requirements for Report: ”The written report should provide a clear documentation about the project’s motivation, the problem addressed by the project, the main technical (AI-related) ideas/concepts behind the project, and the software system design and implementation. The report should describe the experiments performed in the project, and discussions should be included to explain the results of the experiments.”

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 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 traveller’s starting point
2. **States**: Traveller at one of the 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 traveller to the new contact point
5. **Goal State**: The traveller’s destination point

6. **Path**: A sequence of actions from the initial state to the goal state
7. **Path Costs**: The time 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 : Searching for Solutions

3.1 Search Tree

3.1.1 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

3.1.2 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.

3.1.3 Search Strategy

To find the optimal solution we are using the A* algorithm.

FIXME:Or just Dijkstra's/Uniform-cost-search?

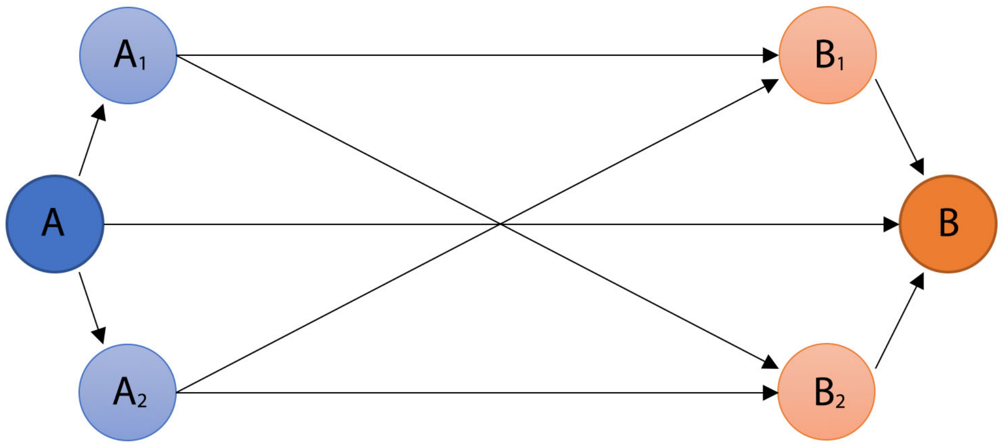
3.2 Constraints

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 $m = 2$.

4 Software System Design and Implementation

To compute the time and price of each edge we are using many different Google APIs.

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



4.1 Object Oriented Design using Java

FIXME:UML Diagrams

4.2 APIs

- 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

4.3 Caching with Bluemix

Some of the APIs are quite slow, especially the QPX Express Airfare API to find flights. To make our path search faster, we cache the information of roads and flights into two databases using Bluemix, one for the road data one for the flight data. This way we can access the databases before using the slow APIs.

5 Experimentation

6 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.