6-14-2019

Trippit: An Optimal Itinerary Generator

Andrew Nguyen

Osama Shoubber

SANTA CLARA UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

Date: June 12, 2019

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Andrew Nguyen
Osama Shoubber

ENTITLED

Trippit: An Optimal Itinerary Generator

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

B. Denfeld
Thesis Advisor

Department Chair

Trippit: An Optimal Itinerary Generator

by

Andrew Nguyen
Osama Shoubber

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Science and Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 14, 2019

Trippit: An Optimal Itinerary Generator

Andrew Nguyen
Osama Shoubber

Department of Computer Engineering
Santa Clara University
June 14, 2019

ABSTRACT

Travelers often lose interest and joy when traveling in tourist-packed areas around the world. As more restaurants and attractions open up in popular cities, the wait and travel time from one location to another inevitably increases. Each attraction has certain hours throughout the day where visitors surge and the wait times increase. In addition, traffic and travel time is an important factor to consider when looking to optimize ones trip. However, with large amounts of attractions, it is difficult to calculate and consider the most optimal routes and times an individual should use to visit each possible attraction. Travelers ultimately face an issue with maximizing productivity for their trips. Our goal is to create a mobile application that utilizes the data from the Google Directions API and Foursquare API to produce an optimal itinerary for travelers to use. Travelers will be able to input their place of stay, attractions they want to visit at their preferred times, and other time constraints to produce an itinerary that will allow the tourist to visit each attraction they please. The Optimal Itinerary Generator will eliminate blind spots in travel planning and as a result, make vacation trips more time efficient and enjoyable.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

As the population in popular cities continues to grow, optimizing our trips and vacations to make the best use of our time is becoming increasingly difficult. In addition to large populations, tourists flock to these cities, further complicating the perfect, productive travel itineraries they worked so hard on. Moreover, the waiting times for restaurants, public attractions, and traffic will inevitably use up valuable time and lessen the experience and immersion. Hence, travelers, vacationers, and tourists face the issue of reliably maximizing their productivity during their trips.

1.2 Background

Two applications that many tourists often use are Yelp and Google Maps. The purpose of Yelp is to provide details about nearby attractions and restaurants such as customer and visitor reviews, price range, and average time spent at the attraction. Also, tourists can use Yelp to discover new places to eat and visit with many reviews, making Yelp an essential application for the typical tourist. On the other hand, Google Maps allows tourists to travel to their destination in a timely manner by always providing the most optimal route. Without Google Maps, tourists would struggle to get from one destination to another, causing them to get lost, and potentially ruin their vacation all together. Hence, Yelp and Google Maps are essential applications that make touring new places more fun and adventurous.

Although the details provided on these applications are useful, they do not provide sufficient detail on the best times to visit particular places. Yelp will only provide the user the average time spent at a particular place. Tourists who want to optimize their trip and spend as little time waiting at a restaurant will need to use Google to look up the times where the attraction or restaurant is most busy. Although Yelp and Google Maps provide a platform to guide tourists of approximate travel and time spent for particular restaurants and attractions, the applications do not inform the user the best possible times to visit the attractions and how long the wait at a restaurant will be. Tourists need to manually search up the typical wait times and time spent at a particular place to determine the best times to visit those places. Furthermore, tourists must also manually check the travel times to and from each destination beforehand to

calculate their best possible options. As a result, with a large number of places that tourists want to visit throughout the week, optimizing the trip becomes extremely tedious and inefficient. Moreover, other applications such as Kayak and PlanChat only allow for users to create itineraries, not optimize them.

1.3 Solution

For better travel efficiency, we created an application that will optimize the entire trip for time. The user will be able to provide information on the different places they may want to visit and the specific time of day that the user may want to visit the place. The application will consider the distance and travel time between each location, as well as the typical wait time and time spent at each location, to determine the most optimal itinerary for the tourist that maximizes efficiency. The application will also support a friendly user-interface that will make travel planning feel more pleasant.

By using the already known data from Foursquare and Google Maps, the Optimal Itinerary Generator will eliminate blind spots in travel planning. We will be using a greedy orienteering algorithm as well as dynamic programming to maximize vacation efficiency, therefore making the trip most enjoyable.

Chapter 2

Requirements

The functional and nonfunctional requirements below define the goals of the project outlined in the introduction. The functional requirements are the technical features that we must implement to define the scope of our project. The non-functional requirements are the features that will enhance the user experience and will allow our project to be easily scalable. The requirements are categorized into critical, recommended, and suggested. It is crucial to implement the critical requirements which are the main functions and tools in our project. The recommended functional requirements will add on to the projects functions and the suggested functional requirements will be integrated if we have successfully completed the critical and recommended criteria. We must adhere our project to the design constraints.

2.1 Functional

2.1.1 *Critical*

- Provide an optimized travel itinerary
- Provide departure time
- Add attractions to intinerary

2.1.2 *Recommended*

- Provide several optimized travel itineraries
- Users can provide several visiting time options for each attraction they add

2.1.3 *Suggested*

- Provide a map directing the travelers trip
- Provide itineraries for multiple trips at multiple locations

2.2 Nonfunctional

2.2.1 *Critical*

- Secure
- Include a notable user-interface to increase user experience

2.2.2 *Recommended*

- Efficient phone storage
- Fast response and always working
- Reliable

2.2.3 *Suggested*

- An elegant application logo/icon to enhance product marketing

2.3 Design Constraints

- The application should be easily accessible and intuitive to use
- The application must be scalable and be able to process large amounts of users and data provided from the APIs
- The application must be accessible on multiple versions of iOS

2.4 Requirements Justification

Because we have about four months for this project, we prioritized the most critical functional requirements as the scope of our project. Completing the critical requirements indicate a working product and a proof of concept which entails scalability. After completing the critical requirements entails the recommended and suggested requirements. The recommended and suggested requirements are the added additional features that help scale our project which will provide the users with more options to increase the efficiency of the product. Lastly, we decided to prioritize our project to be design oriented, such as optimizing time and space complexity and providing a notable user-interface, since modern day products are driven by design.[1]

Chapter 3

Use Cases

Figure 3.1 below depicts how the user will be involved in the system. The application user will be able to add their place of stay during the trip which will denote where the itinerary will begin and end. Then, the user can add and remove attractions to and from the itinerary as well as specify the visit time for each attraction. The user will also be able to add time constraints for their trip. After providing the following information, the user will be able to create their travel itinerary.

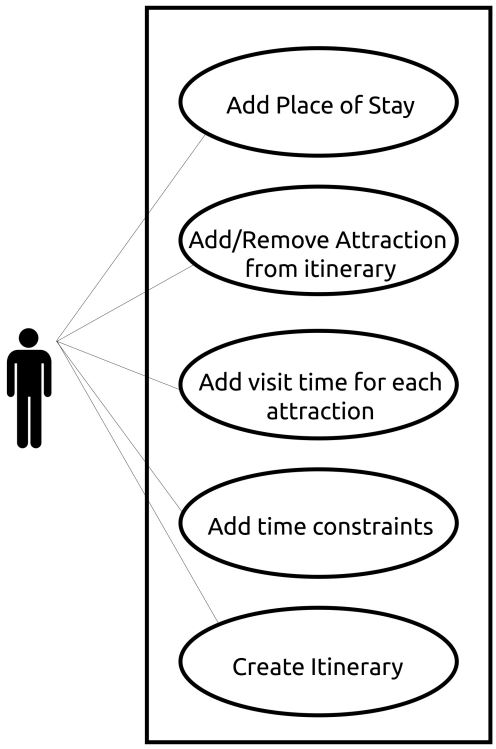

Figure 3.1: Use Cases

3.1 Adding place of stay

Goal: User provides starting point for the itinerary

Actors: Application User

Precondition: the user starts the application

Postconditions: a location of stay is added which serves as the initial starting and ending point of the itinerary

3.2 Adding and Removing Attractions

Goal: User can add and remove attractions from their itinerary

Actors: Application User

Precondition: User provided a place of stay

Postconditions: The user has a list of attractions for their itinerary

3.3 Adding visit time for attrations

Goal: User can add the preferred visiting times for each attractions

Actors: Application user

Precondition: the user added the specified attraction

Postconditions: the user included a specific visiting time constraint to the attraction

3.4 Adding time constraints

Goal: User can add time constraints throughout their day(s) indicating when they are busy

Actors: Application user

Precondition: The user provided a place of stay

Postconditions: The user included time block(s) throughout their trip for the itinerary

3.5 Create Itinerary

Goal: User instantiates an itinerary

Actors: Application user

Precondition: The user provides a list of attractions with visiting times and time constraints

Postconditions: The user can see their optimized travel itinerary

Chapter 4

Activity Diagram

The general work-flow of the application user in Figure 4.1 below has been graphically represented with an activity diagram. The application user must first enter a place of stay and the duration of stay. After, the user has the option to include time constraints during their trip. After providing this information, the user is then prompted to provide several attractions and time of visit for each attraction. The user can remove attractions they have already added as well as edit time constraints. After inputting some attraction(s) and constraint(s), the user can then create the itinerary. Throughout each module, the user will be able to edit their list of attractions and time constraints.

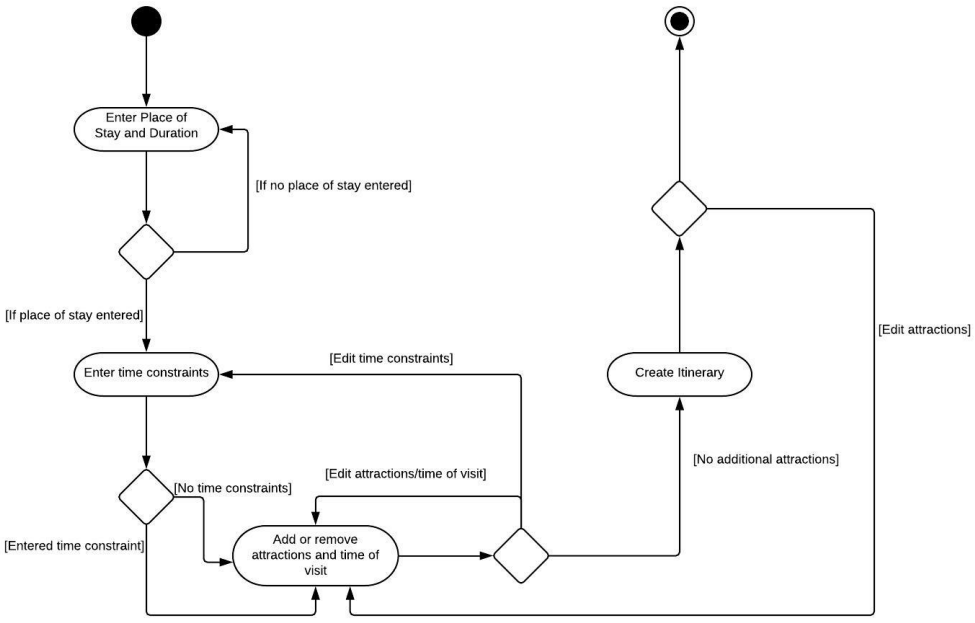

Figure 4.1: Application User Activity Diagram

Chapter 5

User Interface

Our application provides a seamless user interface with different displays for different user actions. The user will be able to navigate between each screen on the application with a navigation bar at the bottom of the screen. Figure 5.1 depicts the expected user-interface. From left to right, the modules are: initial start screen, adding a place of stay and time constraints, list of nearby attractions and adding attraction, and the generated itinerary.

The application begins with a seamless splash screen that suggests to the user the purpose of the application. The user begins the application by entering their place of stay, such as their Airbnb or Hotel location and preferred starting time for the itinerary. The location entered will be the starting and ending location of the itinerary. The user can them enter several attractions to their itinerary. After the desired attractions are entered, the user can them produce the optimal itinerary with a single click of a button. The user would then be prompted to the final screen, which is the itinerary screen, as displayed in figure 5.1.

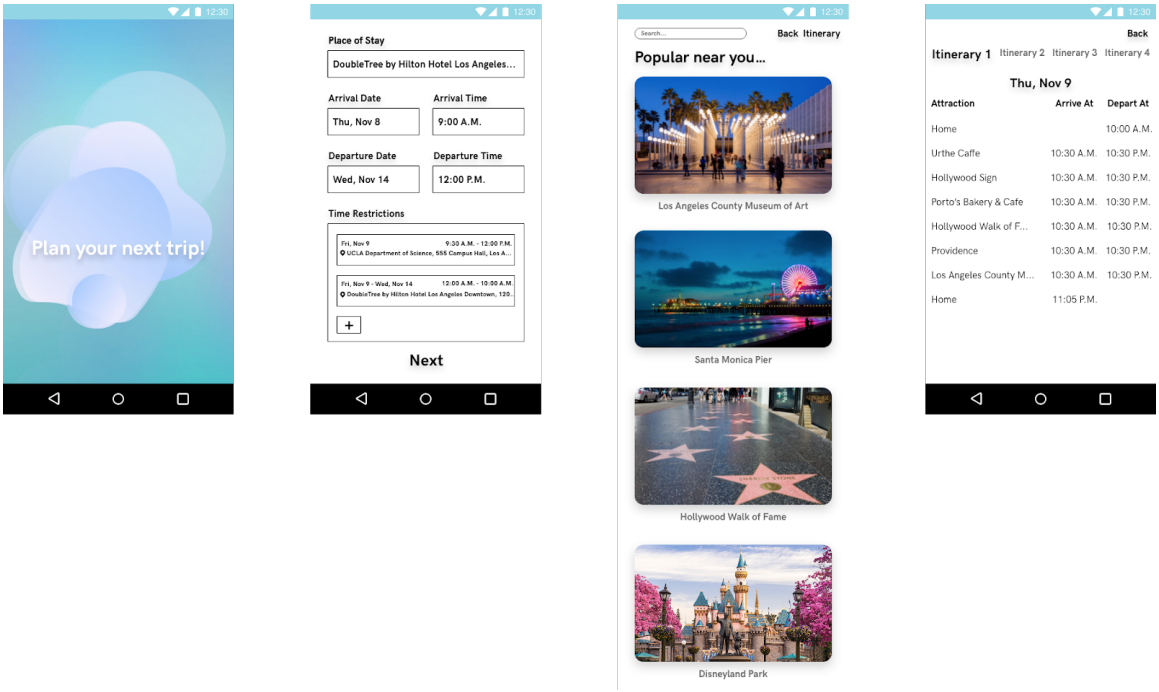

Figure 5.1: User Interface Prototype

Chapter 6

Technologies Used

Below is a list of technologies we used. As the system was implemented as a web application, we predominantly used web tools for development.

- React Native 0.57 framework
- Foursquare Places API
- Google Directions API
- Google Places API
- Native-Base
- Github
- Adobe XD/PS/AI

Because we are creating a mobile-application Android based, we will be using React-Native as or main programming language for our application development. React-Native offers simultaneous Android and iOS development which will help expand the user pool. The programming language also offers several open source User-Interface libraries, such as Native-Base which we used to help enhance the user experience of our application.[2]

The data used in our application will be provided through Google Distance Matrix, Foursquare Places API, and Google Places API. We first used the Google Places API to provide a text autocomplete feature for the user. This allows the user to properly type in the attraction name, mitigating the errors for the user. This also allows for the back-end portion of the software to send proper API requests to the Google Distance Matrix API and Foursquare API.

The Google Distance Matrix provides the travel times including traffic hour from one point to another, which is an essential aspect of optimizing our itinerary. The back-end is given a list of attractions provided by the user-input from the front end of the software. The back-end then makes the API call to Google Distance Matrix API to receive a

duration matrix from each location. This duration matrix represents a directed graph from each attraction, represented as nodes.

On the other hand, Foursquare Places API provides the expected wait and time spent at each attraction, which is also essential to the development of our application. The back-end portion of our software sends the same list of attractions to the Foursquare Places API to retrieve the popular hours and average time spent at each location. The data from these APIs were provided in JSON format, so we were able to easily reference the objects in simple Javascript syntax.

Figure 6.1 in the following page showcases how we will use the API to retrieve and compute the data to produce the optimized itinerary. The list of places and popular hours are retrieved from Google Places API and Foursquare Places API from the user input. The list is then sent over to our algorithm processing to fetch the travel durations from Google Distance Matrix API. The travel durations is used in conjuction with our Touring and Traveling Salesman Algorithm to produce the optimized itinerary which is sent back to the user.

We will use GitHub for version control of previous work. GitHub will allow multiple members of the team to work on the project simultaneously, increasing productivity. Lastly, we will use Adobe XD, Photoshop, and Illustrator to prototype the user interface, design the application logo, and create widgets for the application.

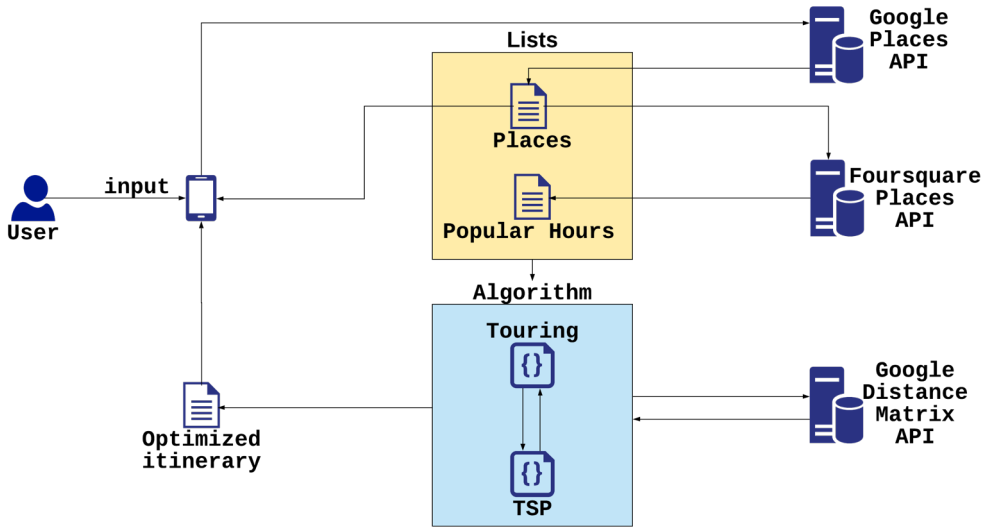

Figure 6.1: API Usage Architecture

Chapter 7

Architectural Diagram

Figure 7.1 on the following page depicts the architectural diagram of our project. The architecture of our system is based on a client that connects to a server which accesses the API data. The connections will be implemented as a mobile application through an API call request.

We will be using a **Client-Server Architecture** since we will be processing and accessing data from Google Distance Matrix API, Google Places API, and Foursquare Places API. All of the relevant data needed to compute the travel itinerary are provided with these APIs listed, so there is no need to use a databse. In addition, we will not be storing the information from the API in a database, since the metadata will be too large due and the consistent change in travel times. Hence, the Client-Server Architecture will be most adequate for our system, since users can freely use the application independently. In addition, changes in traffic and wait time to and from attractions will be updated in the API frequently, so the Client-Server Architecture will be able to provide itineraries that are up-to-date.

Currently, our algorithm computes the itinerary in O(n-1)! factorial run-time. This is because our solution builds off the Traveling Salesman Problem which is known to be O(n-1)! run-time. Our algorithm and performs its computations in the phone processor. As a result, a limitation in our project is that only up to 12 itineraries can be produced under a reasonable time of under 30 seconds. Anymore additional attractions added to the list will either take too long to compute or will cause the application to crash. This is because the phone processor cannot handle factorial computations for the single application.

Nonetheless, There are currently software technologies such as Google OR-tools that approximate the traveling salesman problem to be more efficient in run-time. However, Google OR-tools is incompatible with React-Native. Hence, a more ideal design architecture would be to use an external server which the client would connect to. The server would then make the API calls and process the data received utilizing Google OR-tools. Processing and computing the data on the server would be much more efficient, since the server would be able to compute data faster and handle large amounts of data. Figure 7.2 on the following page depicts the ideal server architectural diagram for our project.

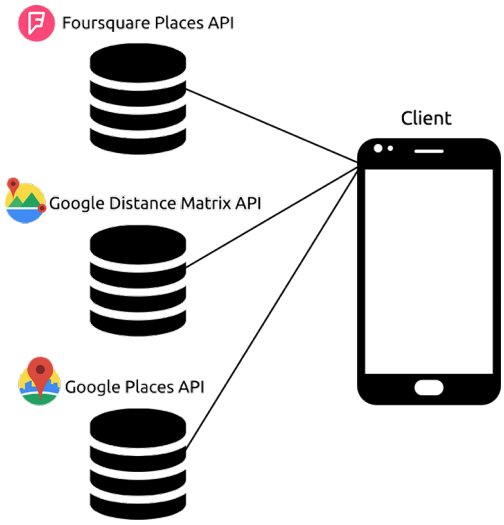

Figure 7.1: Client-Server Architecture

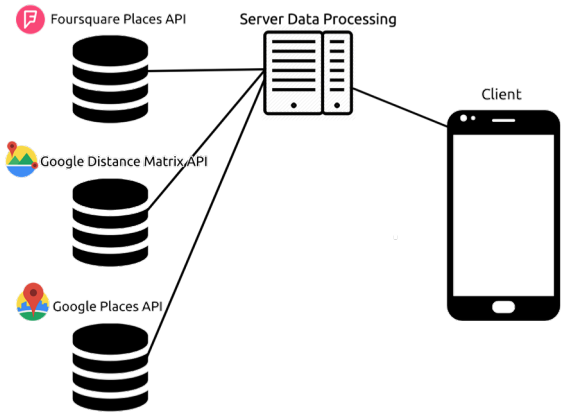

Figure 7.2: Ideal Client-Server Architecture

We are aware that we have a single point of failure. If the API servers crash, then our application no longer works. Fortunately, the metadata provided in the API can be downloaded in XML format. In the case of the API servers crashing, the metadata from the XML format will still be able to provide a working product, but the itineraries provided may not be as optimal.

Chapter 8

Development Process

We first began our development process by revisiting the Traveling Salesman Problem. Our project utilizes the infamous algorithm which is an NP-hard problem. However, our solution turns the algorithm into an NP-complete problem. This is because the Traveling Salesman Problem assumes that the distance between two points is the same in an undirected graph. However, under a real world setting, the distance between one point to another and back may be different due to traffic or other underlying conditions. Hence, our project deals with an asymmetric Traveling Salesman Problem.

8.1 High-Level Pseudocode

Figure 8.1 below depicts the high-level pseudocode for our itinerary calculation, where G denotes a graph containing a set of vertices, V, B denoting the budget or amount of time the user has in the day to spend, and s denoting the user's starting location. The algorithm calculates the traveling salesman problem for each iteration of the vertices provided and calculates the best margin for each iteration. The best margin determines the most optimal itinerary considering the ratio between the *utility*, how long the user spends at each attraction, and *cost*, how long the user takes to travel between each location, to maximize the itinerary's potential. This ratio best fits an ideal user's goals, since travelers typically seek to get the best "bang for buck" when traveling. The run-time for this algorithm is O(n - 1)!, which is factorial time. There are currently approximation tools for Traveling Salesman such as Google OR-Tools which can reduce the run-time of our algorithm. However, Google OR-Tools is currently incompatible with React-Native. Given our project time constraint, it was best to provide a proof-of-concept, which is what this algorithm offers.[3]

```
Tour (G, B, s):
    P* = s
    do:
        P = P*
        best_margin = 0;
        P* = null;
        for each v : V {
            P' = TSP (vertex(P) union v)
            margin = (utility(P') - utility(P)) / (cost(P') - cost(P))
            if (margin > best_margin and cost(p') < B)
                best_margin = margin
                P* = P'
        }
    while (P* != null)

    ts = 0
    for each edge (u,v) : E(P)
        tv = currentTime(u) + durationTime(u) + travelCost(edge)

    return (P, t)
```

Figure 8.1: High-Level Pseudocode

8.2 Front-End Development

The front-end development consists of the screen display, screen navigation, buttons, and text input which encompasses the User-Interface development. We utilized Native-Base which is a React-Native User Interface library for most of the Front-End development.[4] We implemented four screens total. The first and opening screen is the home page which informs the users of the functions of the application. There is a navigation bar at the bottom of the screen so the user can navigate between the screens. The second screen is where the user inputs their list of attractions. This screen supports the Google Places API auto-complete feature. The third screen displays the list of attractions the user has input. At the top of this screen, the user is able to select their preferred start time. At the bottom of the screen, there is a button that allows the user to generate their itinerary. Because our project was heavily algorithmic oriented, most of our development process was in the Back-End.

8.3 Back-End Development

The back-end development consisted of performing the API calls, computations, and processing. After the user inputs an attraction in the screen display, the front-end sends the input to the back-end to make API calls to the Foursquare Places API and Google Distance Matrix API. The Foursquare Places API fetches information about the opening and closing hours of the attraction, average time spent at each attraction, and popular hours at each attraction. This data is utilized for our algorithm computing. The popular hours and list of attractions are used with the the modified asymmetric Traveling Salesman problem to produce the optimized itinerary. Ultimately, the back-end processes the data and computes the itinerary through the algorithm with the data from the APIs.

8.4 Development Challenges

There were several challenging encounters throughout the development process. First off, the project was algorithmically challenging, requiring an implementation of an asymmetric Travel Salesman algorithm. Without using Google OR-Tools, we had to use a brute-force implementation of the algorithm. Initially, we developed our algorithm recursively. However, React-Native's asynchronous nature does not allow the team to utilize a recursive implementation. The team resolved this issue by implementing the algorithm iteratively and to make use of Promise functions.

React-Native's asynchronous nature also imposed other issues in our development process. Whenever the application makes an API calls, by the time data is returned, the User-Interface has already finished rendering, so the data cannot be displayed. The Promise function is a multithreaded function that awaits a resolve or rejection. After execution, the promise can either fail, where the program can then handle the error, or once it resolves, the program can continue with the next action.

Moreover, react-native asynchronous nature also meant that data is not stored globally on mobile applications by

default. As a result, copies of data must be sent from screen to screen to be manually synchronized. The team resolved this issue by saving the state variables which can be changed at run time to induce rendering

Chapter 9

Design Rationale

Like Yelp and Google Maps, we decided to make our project a mobile application to increase the number of potential users. Tourists often travel by flight and by car, and mobile applications will make this information easily accessible when out of the house.

To develop our application, we decided to use React-Native to program for iOS and Android simultaneously. React-Native development is open to all operating systems [5]. Pure iOS development requires using XCode on a Mac and not all members of the group own a Mac. React-native also provides fast build times and reloading[4]. Given the time-constraint and the team members' past experiences with JavaScript, programming with React-Native was in the groups best interest to increase development productivity.

Chapter 10

Test Plan

10.1 Primary Test

We began our initial testing after developing the travel optimization algorithm. We began with small test cases such as providing a small list of travel visits and checking our result manually with the metadata provided to us from the API. First, we began testing the Google Places API autocomplete feature to ensure we were getting the proper attraction names. Then, we tested our Foursquare API calls utilizing the list we produced. Lastly, we tested the distance matrix provided from Google Directions Matrix API. We included edge cases such as a larger list of travel visits to determine when we should catch an exception error. Because all of the computation is on the phone, our application can only handle up to 12 inputs. The

10.2 Unit and Integration Testing

After ensuring that our algorithm is accurate and provides the correct results, we tested the individual functions with white box testing. Each functions of our application such as adding an item and producing an itinerary must access and retrieve the correct data every time from Google Places API, Google Distance Matrix API, and Foursquare Places API. We tested these constraints by feeding our application different kinds of information, ensuring to cover corner cases, like misspelled location names. However, because of the Google Places API auto-complete feature, we mitigate the chances that users misspell the location name. After finishing unit testing the group began integration testing, where the individual modules were combined and tested as a whole. We tested the user inputs and fed the inputs to our algorithm and API calls to ensure we were getting the proper results back.

10.3 Alpha Testing

After finishing the unit and integration testing, we began alpha testing with 10 users. We sampled data from a few users that showcase our alpha testing results.

Figure 10.1 in the following page depicts the fully optimized itinerary person A produced that begins and ends at UCLA. The itinerary provides the time it takes to travel to and from each attraction. The itinerary also provides the user with the average time spent at each attraction.

Figure 10.2 in the following page illustrates the results from a satisfaction survey conducted with the alpha testers. In terms of satisfaction, the alpha users were mostly satisfied with their experience with out application. This feedback allows the group to understand the progress of the project to improve and add further development.

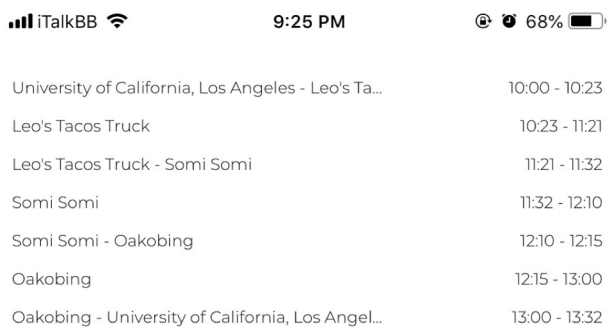

Figure 10.1: Person A - Itinerary Result

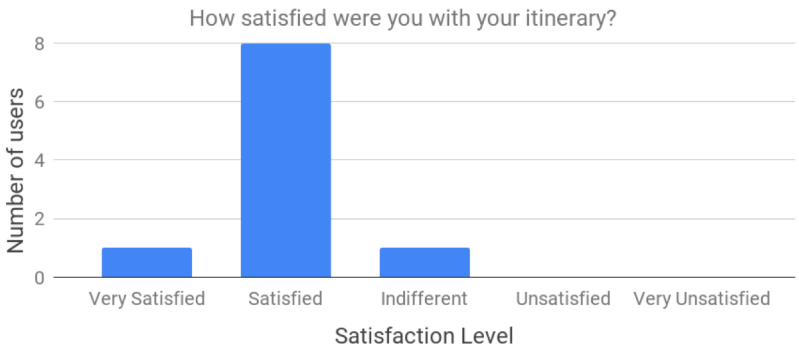

Figure 10.2: User Satisfaction Survey

Chapter 11

Risk Analysis

Table 11.1 in the following page highlights the potential risks and that may have occurred during the project. Each risk contains the effects on the team, probability of occurring, solutions to mitigate the risk, and the duration of the risk. Overall, the biggest obstacle of the team was the learning curve. We had insufficient understanding of the technologies needed to implement our application in the beginning. Mobile application operating systems are a lot different than the traditional Mac OS and Linux environments that we are used to developing on. Hence, learning and understanding the technologies was a critical part of the project development process. As a result, bugs became an even bigger issue than before. Because React-Native is a relatively new language, there aren't as many online resources to seek for help as compared to other common languages like C and C++. React-Native has also undergone changes where some library functions were deprecated that we could no longer use. Ultimately, thorough research and understanding of the programming language and mobile phone operating system were critical for the project success.

Table 11.1: Risk Analysis Table

Risk	Effect	Probability	Severity	Impact	Solution	Prevent
Insufficient Understanding of technology	Slowing down the development process	0.5	7	3.5	Explore resources online for help (stackoverflow, github)	Learn Android Studio ahead of time (Andrew).
API servers crash	Loss of essential data to our project. Our project can't work without the APIs.	0.000001	10	0.00001	Use previously saved XML metadata to temporarily process itineraries.	Save several samples of XML data from the API servers.
Unoptimized Itinerary Generating Algorithm	Loss of scope of the project. It is essential that the algorithm is optimal.	0.05	10	0.5	This is not an option. We will seek help from peers and professors to ensure that we complete the algorithm.	Start working on the algorithm before the development process. Check and show professor mathematical proof of optimized algorithm.
Incompatible application on different android devices	Not meeting design constraint. Fewer users.	0.1	3	0.3	Update code to be compatible with Android devices and most current version.	Read the guidelines in the open source libraries to ensure the android development is up to date.
Application crashing from processing large amounts of metadata	Limits the application potential.	0.2	8	1.6	Debug crash by catching exceptionDisplaying error message instead of crashing.	Write catch exceptions and test cases throughout the development process.
Loss of Code/ Unsaved Work	Slowing down the development process	0.001	8	0.008	Restart work and frequently save work/push code onto Github	Frequently save work every few lines of code. Push code onto github frequent.
Bugs	Project development obstacle, preventing further development until fixed	1	5	5	Allocate time to debug as well as development. Seek online platforms like Stackoverflow and advisor for guidance	Comment and write clean code to prevent bugs. Test functions through development process.
Bugs	Project development obstacle, preventing further development until fixed	1	5	5	Allocate time to debug as well as development. Seek online platforms like Stackoverflow and advisor for guidance	Comment and write clean code to prevent bugs. Test functions through development process.

Chapter 12

Development Timeline

Our project design and implementation was a five month process. We began our implementation at the beginning of Winter Quarter. We broke down our implementation process into a 14 week timeline. We spent the first two weeks developing the algorithm, two weeks learning the basics of the technologies we were using, six weeks implementing our APIs to work in conjunction with each other two weeks on implementing our algorithms, and the last two weeks for User-Interface development and testing. Figure 12.1 depicts the process of our project design and implementation from start to end.

The gantt chart showcases the strict deadlines the team needed to follow to achieve optimal results in our project. The team also met weekly to work on the design project and to pitch ideas on how to overcome the challenges each team member faced.

Task Name	Duration	Start	Finish
UI/UX Mockup	5d	11/12/18	11/16/18
UI/UX Prototype Finalized	1d	11/23/18	11/23/18
Developing Optimization Algorithn	45d	11/26/18	01/25/19
Android App Development (functic	20d	01/28/19	02/22/19
Android App Development (UI/UX	15d	02/25/19	03/15/19
Prototype Completed	1d	03/15/19	03/15/19
Testing/Debugging	31d	03/15/19	04/26/19
Deploy	5d	04/29/19	05/03/19

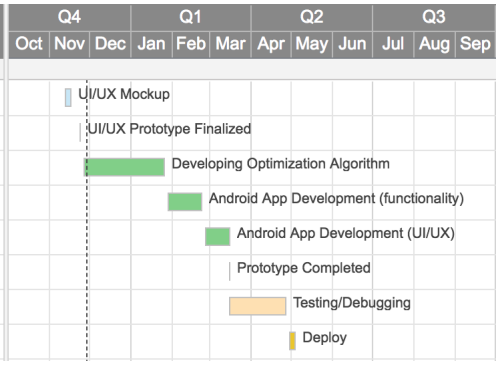

Figure 12.1: Project Development Gantt Chart

Chapter 13

Societal Issues

The functional and nonfunctional requirements below define the goals of the project outlined in the introduction. The functional requirements are the technical features that we must implement to define the scope of our project. The non-functional requirements are the features that will enhance the user experience and will allow our project to be easily scalable. The requirements are categorized into critical, recommended, and suggested. It is crucial to implement the critical requirements which are the main functions and tools in our project. The recommended functional requirements will add on to the projects functions and the suggested functional requirements will be integrated if we have successfully completed the critical and recommended criteria. We must adhere our project to the design constraints.

13.1 Ethical

The key ethical question that came to mind when creating this project was how the users itinerary data would be stored. If a hacker could somehow find someones itinerary, they could figure out where they will be at any given time of the day. However, our application did not store any data inputted by the user, it only uses their preferred departure time, and the rest of the computations are done by secure APIs that are managed by companies like Google who have reliable and rigid security.

13.2 Societal

Since our product can recommend attractions and also generate itineraries, it may impact travel the job security of travel agents. In a way, our project can be seen as automating the job of travel agents. However travel agencies provide additional services and partnerships, which our application does not. Therefore, it is safe to say that we are not endangering the lifestyles of this individuals in this profession.

13.3 Political

Our project does not incorporate political elements. We use public data provided by Google and Foursquare. Because we do not store user data, we do not have any security liability with our users. Hence, our project does not encompass or address any political implications.

13.4 Economic

Since our project is a mobile application, it did not have any costs associated with it other than the materials and devices needed to create it. In its current stage, the application does not interface with any of our own servers, but it utilizes services from other companies. We did not need to pay for these services since they are free given a small number of uses. Since we were testing, our calls to these API calls were very minimal; however, if our project were to scale up accommodate the masses, we would need to pay fees for the API calls since we would be sending tens, hundreds, or even thousands of calls to the servers. In addition, one of the future goals for this project is to process all the computations on a server. Products like Amazon AWS, Microsoft Azure, and Google Cloud provide servers for developers to use for their applications at a fixed cost. All in all, our project does not require substantial amounts of money for deployment.

13.5 Health and Safety

When traveling, travelers are responsible for their health and safety. The itinerary we provide to the users do not affect their health and safety.

13.6 Manufacturability

Because our project is software based, our product can be downloaded on any iPhone with iOS versions 12.0.1 - 12.2, which is the most recent versions of iOS software.

13.7 Health and Safety

When traveling, travelers are responsible for their health and safety. The itinerary we provide to the users do not affect their health and safety.

13.8 Sustainability

Our project can be sustained indefinitely and may actually become more important over time. As populations continue to grow, people may find that optimizing an itinerary is crucial to having an enjoyable experience. Furthermore,

our product would not have to be maintained nor would it deteriorate over time.

In a broader sense, our product may become more sustainable over time as phone processors and servers are able to compute more with less energy consumption.

13.9 Environmental Impact

Since our products only resource are the servers it would run on, it is important to consider the effects of keeping server clusters at stable temperatures. Server clusters can contain tens of thousands of clusters and it takes a lot of energy and therefore, resources, to run them. The best that can be done on our end is to make sure our server back-end runs efficiently as possible to minimize processing time.

13.10 Usability

One of the goals for our project was for it to have a friendly and elegant user interface. From our alpha testing results, our testers reported they were mostly satisfied with the user experience. In addition, our application is compatible with the iPhone voice recognition which allows for accessibility for users with disabilities. Despite the positive results, we look forward to further improving the usability of our application.

13.11 Lifelong learning

This project definitely inspired us to learn newer and emerging technologies in our field. React Native, the platform we used to build our product, is less than four years old. Most applications that are developed are all done natively on platforms like Swift and Android Studio. However, React Native is becoming more popular, just like its familiar, React. By integrating React components with native code for iOS and Android, Facebook, the creator of this platform, has pioneered the future of mobile application development. This project was an incredible learning experience and an excellent insight to the industry.

13.12 Compassion

A simple fact of life is that you will never get back the time you spend. Every second matters in our limited time on Earth. With time being so valuable, we must seek ways to maximize our efficiency and potentially enjoyment of our time. Although being stuck in traffic or a long line for hours is not the greatest suffering imaginable, we would much rather spend that time doing the things we love. Our project sought to relieve people of the unnecessary stress that is associated with traveling with minimal effort from the user. Efficiently planning your itinerary could easily make a trip much more enjoyable, relaxing, and stress-free.

Chapter 14

Conclusion

There is currently no method of optimizing an itinerary for time. As a result traveling has become less efficient and less enjoyable. However, by utilizing a variation of the Traveling Salesman algorithm alongside the data from the Google Distance Matrix Foursquare Places API, optimizing one's travel itinerary is possible. The asymmetric Traveling Salesman problem is still classified as an NP-complete problem, with our algorithm running in factorial run-time. Nonetheless, under real-world conditions, users would typically have around up to fifteen attractions in their itinerary per day. With powerful severs and approximation tools such as Google OR-tools, the computation would take a matter of seconds.

Under an ideal or company setting, it is best to utilize a powerful server that can handle large amounts computations for our application to scale. However, the prototype we produced proves that itinerary optimization is feasible. With Trippit, users can spend every minute of their vacation hassle-free.

Bibliography

[1] Evan Hanson. Ux design: The importance of user experience. https://www.rocket55.com/ux-design-the-importance-of-user-experience/, March 2017.

[2] Native Base. Native base. https://nativebase.io/. Accessed on: March 28, 2019.

[3] Kostas Kolias. Orienteering algorithms for generating travel itineraries. https://www.math.uwaterloo.ca/~cswamy/papers/trips-wsdmfnl.pdf.

[4] Cifuentes Jamie. Android vs the iphone: which is more expensive?about the android open source project, February 2013. Accessed on: November 29, 2018.

[5] Android. About the android open source project. https://www.mockplus.com/blog/post/is-android-better-than-ios. Accessed on: November 26, 2018.