ICSI499 Capstone Project Report

FedEx Stop Routing

Team - 05 - Technocrats

Charles Porter (001261651)
Thomas Bridges (000919450)
Joel Bernhardt (001165470)
.....
College of Engineering and Applied Sciences
University at Albany, SUNY

Project Sponsor

Aimee Witko
FedEx Express
590 Broadways,
Menands, NY, 12204

01-12-2020

Acknowledgements

A special thank you to the guidance and supervision provided by Aimee Witko and everyone at FedEx Express.

Abstract

This project calls upon our team to create a solution for an issue that almost every driver faces at a company like FedEx. “What is the most efficient route for me to take in order to reach every stop, while also traveling the shortest distance possible.” A solution for this problem has the potential to save corporations hundreds of thousands in gas and labor costs. We tackle this problem by creating an application that can operate on the android devices provided to the drivers. Once launched the driver will login with their driver ID and password stored locally in a sqlite database. Once properly authenticated, their stops for the day are uploaded to the maps activity layer, and presented to the driver in the form of pins on a map. The driver can then click “sort”, where all of his/her stops for the day are ordered, and a path is created via the simulated annealing algorithm. Stops 1-15 will then be presented to the driver, starting with the one closest to them. They can also select “show all”, and the entire path will be shown, along with every stop. “Next stop” will simply delete their current stop, and add one to the current 15 being displayed (i.e. stops 2-16). Any note associated with a given stop is also displayed to the driver. Where an initial baseline distance is created by measuring the distance from one stop to the next in random order for all stops in the dataset, an additional “final” distance is created from one stop to the next once the path is created. When comparing these two distances, we can see that the distance traveled along our created path is shorter than the distance traveled given a random path. It is important to recognize however, that these values are yielded from our dataset of stops, and are subject to change given any other dataset.

Contents

1.Introduction

Our team, team number 5, the Technocrats, picked an overall interesting, and complicated problem for us to solve, as part of our graduation into real world programming, and developed an application for FedEx. The problem faced was that FedEx drivers currently do not use, and do not have access to a cohesive way to organize stops into a proper route. The current solution involves printed maps with no cohesive route. With no clear cut routing solution, packages may be late, which not only affects customer satisfaction, but a monetary loss for FedEx, as they have a policy where late deliveries result in the full refund of shipping.

This problems' difficulty is due to a number of reasons, and features a number of limitations and complications. The primary difficulty is in the actual equation and computing of a route. Calculating this route is more commonly referred to as the "Traveling Salesman problem", which is widely considered to be one of the most famous, and long standing problems to solve in computer programming. (http://www.math.uwaterloo.ca/tsp/problem/index.html)

This specific calculation is one of the most complicated currently proposed, due to the number of ways in which it can be solved, and the currently required number of calculations that must be performed, especially with a higher number of stops required, and worsened by every additional requirement to the stop routing methodology. Effectively, every time a stop is added, the overall number of calculations to "brute force" is added at an almost exponential fashion, getting worse and worse, with each added stop.

To make things worse, we have an additional problem that must be solved. Even with this enormous number of calculations that need to be performed, they all must be from within an android application running on a generic android phone, with limited computational power. To further complicate this, FedEx additionally has priority deliveries, to even further cause complications. Additionally, as the cherry on top, some days, drivers may deliver upwards of 100 packages or more, which draws out this calculation in an obscene and immense manner.

Some similar applications do exist, which can create a route. But they fail to account for issues such as number of stops, requirement for being android based, amongst others. By being android based, allowing users to skip stops, and to calculate for the number of required stops, our application is aiming to surpass others currently in development and use.

2. Background and Related Work

Some existing routing solutions do exist, for example, google does have an API for route calculations (https://developers.google.com/optimization/routing/tsp), But this example lacks certain elements, like being unable to calculate a route over 100 stops. It also lacks the ability to add stop priorities, which is important to FedEx. Other solutions exist in various forms, even in the form of a mobile app, but even if they were to take all these factors into account, there is one unaccounted factor that these are all detracted by. A company the size of FedEx would prefer all software to be owned by FedEx and built in house, without any outside dependencies.

The core idea behind our solution is that we should develop a mobile application that should be easily usable for the FedEx drivers. This application should contain all of the core functions that drivers might need, from routing, to showing an organized group of delivery locations, to having a simple to use login, and any other features that drivers would find useful. It is also absolutely critical that this does not hinder them in their duties, and is not too slow or cumbersome, as those would be serious issues.

Our solution is novel for a number of reasons:

1. Easy to use, no formal training required. Does not require any additional knowledge to use, no user input other than login is required.
2. Clearly shows routes and stops, without any clutter or questions.
3. Is not complicated to navigate, no overwhelming options or tools, simple easy to use user interface
4. Usable for multiple drivers, from the same application at login. Routes are assigned to a route not the device or driver and the stops are loaded when the driver enters the route they will be doing that day.
5. Does not rely on any outside services, and could easily be used for an “in house” style of operation.

There are some related, similar programs in existence already. Most everyone is familiar with programs like Google Maps, which has a basic routing function, that we do take advantage of in our program. Below is a list of such programs, and some of the reasons why they may not be ideal for FedEx’s current use case:

MapMarker, CoPilot, etc. - Mobile applications; these are limited in scope to what FedEx would need.
OptimoRoute, Route4Me, Samsara, etc. - There are already solutions out there however if FedEx were to use one of these services they would be reliant on proprietary solutions.

Here are the sources for each of the above listed programs, from their websites::
Optimo - https://optimoroute.com/features/
CoPilot - https://copilotgps.com/en-us/
Mapmarker - https://www.mapmarker.app/
Route4Me - https://www.route4me.com/

As you may have noted from the above list, the programs do not check all the boxes for FedEx's requirements, and critical/common to all of them, is that they are all proprietary solutions, and are not in house programs. FedEx would not have withstood having a critical program, such as a driver routing app, maintained and built by an outside contractor, for a multitude of reasons, including cost, dependencies, and management.

Utilizing an outside contractor will be more expensive, especially as custom additions are required for FedEx. Every time an additional change is required, or new function added, contracts need to be renegotiated, and lawyers paid. It would be much less expensive for FedEx to just have its own small in-house team, which could readily maintain and modify, both adding and removing, requested functionalities.

Additionally, by being in house, all dependencies are handled in house. The systems used can be hand selected, from both inside FedEx, and outside, limiting the number of dependencies it may have. If changes are needed in an area, such as the database that maintains deliveries, it can be handled inside the company, without requiring inter-company work. In the event of catastrophe, there is staff on hand to find where the problem occurs, and how it's to be fixed, and with a limited number of outside services in use, it's much easier to know where the problem is occurring.

3.1 General Guidelines

The idea proposed for this application was to use the simulated annealing algorithm to output a route that is more efficient than the seemingly random order of the initial stops. There is no equation for the algorithm (refer to section 4.3 for an in depth explanation of the algorithm). This is part of the reason why we chose this algorithm. It allowed flexibility and fine tuning (specifically in terms of run time) of the variables to make it work to our needs. For the database SQLite was used. This was used over other database systems because it is built into Android and easily allowed us to make and access a database. Google Maps was used as the main display since there is already a vast amount of functionality that Google has already implemented. We just had to work the Google functionality into our application.

There were many assumptions made, specifically about what can be delivered in the final product. Unfortunately due to time restrictions, lack of knowledge in the functions we wanted to implement, and other school related commitments we were not able to implement everything. A couple examples are more GUI related functions, distinction between priority, and getting the whole application working from a remote server. Almost everything that was designed and implemented was made from scratch with no prior knowledge in Android development. We argue that these restrictions are the reason for not being able to deliver on everything that was promised. In terms of functional requirements, everything was implemented. However, we were not able to implement everything.

3.2 System Requirements

For our application, we plan to have one primary user class, with a handful of requirements, both functional and nonfunctional:

USER CLASS:

All Classes will be available to the Driver user class. This is because the driver will be the only one using this application, and they are the entire reason for the application's existence. The whole goal of the application is to provide the Driver with the tools to see their stops and provide them a line of travel.

FUNCTIONAL REQ:

The Driver Use class has a few functional requirements. The stops need to be loaded into the application. After the stops are loaded, the driver needs to be able to sort those stops into an order that gives them an efficient line of travel. The stops need to then be displayed for the driver to see where they need to go for each stop.

NON FUNCTIONAL REQ:

For non functional requirements, we focus around the concept of CRUD (create, read, update, delete) to maintain functionality. Firstly, the driver should be able to manually add and subtract stops from their list of deliveries, in the event they need to skip a delivery, or pick up a new one, for whatever reason. They should also be able to view information pertaining to each stop, such as stop number, location, address, etc, simply for ease of use.

Secondly, an authentication will be required for each driver. Individual login usernames and passwords are required for each driver, in order to log in to the application, and have access to their route. The driver would login at the beginning of the day, enter their route number and password, and the stops are loaded for them.

OPERATING REQ:

For the operating requirements, our application is intended to be run on an android phone, which has its own specific limitations and requirements. There will also be a database for all of the day's stops. There were plans to make each route have its own database but due to time limitations the information was implemented by use of a text file which simulates each line of a real database where this info would be stored.

DESIGN IMPLEMENTATION:

Some of our design implementation will include functions such as a limit on the number of stops displayed. This is because there shouldn't be a need to show every stop at all times. The application should only show a set number of the closest stops, and this will make the application easier to use since the screen won’t be as cluttered. This will help cut down on user fatigue, as there's less confusion in the presented interface.

Additionally, our application will need to be run on a standard phone. FedEx will provide the same make and model of phone to all drivers using the application. This will limit the need to make many different versions of the application, so that there’s no need to custom tailor each for each version.

The application will also be required to function on the current Android operating system. Considering all drivers are given android devices running the latest operating system, the application must be seamlessly integrated with the hardware/OS available, and not be reliant on any older hardware or software.

3.3 Technical Design

When the driver first opens the application they will be greeted with a login page. This login authorization references a database that stores the username and passwords. When the driver logs in they will see markers of all their stops. The reason to show all the stops at first is to see if there are any stragglers that are not supposed to be on their route or should go to a different driver. When the driver is ready, they can then hit the sort button which will sort the stops that they have, starting with the stop that is closest to the driver.

After the stops are sorted the driver can then fully use the application. The stop the driver has next will be highlighted in green. This is so that they can see at a glance where the next stop is.

The process/application would work as follows: If the driver knows how to get to the stop they can just drive to that stop. When they get to the stop and make the delivery they can then hit Next Stop and the next stop will highlight. The stop they just delivered gets removed and another stop gets added to the map. The map will show at most 15 stops. This is why the last one gets deleted and one more added. Of course the one added is the next one in the list after the last one that is already shown. If at any time the driver wants to see the full map of all the stops left and line of travel they have to take, they can hit the Show All button. When the driver is ready. When the next stop button is pressed, only 15 stops are shown again. If at any point the driver needs directions to a stop they can press the marker that they

want to go to and another button will show up that will take them to google maps and they can use Google Maps navigation to get there. This process repeats until all stops are delivered.

3.4 System Implementation

In order to implement this application many, if not most, of our computer science theories and academic studies were used. The most impactful and crucial parts to this application were the routing algorithm, the database and storing of info needed, and Java programming knowledge as a whole. Android studio and DB Browser were used as the tools to code and implement the application and database respectively.

The whole purpose of this application was to make a route given a data set of stops for a FedEx driver so you can imagine the most important aspect of this project was the sorting algorithm. The sorting algorithm essentially puts the stops in an efficient line of travel to reduce the drive time to a minimum. Many days and hours were spent trying to find the best algorithm for the job. This is a traveling salesman problem, which is an NP-Problem (non-polynomial time complexity). This means there is no “found:” solution yet. The best that can be done is a “good enough” solution. After lots of research into different solutions such as 2-opt, hill climbing, brute force, etc, we finalized on using simulated annealing. This is because given certain variables in the algorithm you can control how long it runs for. This is crucial since time is a factor in our application.

The database keeps track of driver login info. This is of course important because the software and data need to be protected from people outside of the FedEx system. In our case, only a driver would be able to login.

Many aspects of computer science programming were used in this application. Data structures such as arrays to manipulate the stop order, reusable code such adding the markers to the maps with different buttons at different times, code documentation to explain how the algorithm and different functions work, testing with different values and using debugging tools to figure out errors and how to fix them, and last be not least lots and lots of reading of documentation. For example, Google Maps API was used and since no one on the team has used Google Maps API, for everything that was going to be implemented, documentation had to be read.

Android studio was extremely helpful in getting a foundation made for this application. It was started with a Google Maps Activity which generated the starting code for the map. This code was simply an onCreate and onReady method to get the project started. It was also very helpful for making the buttons and providing an easy way to manipulate the look and feel. DB Browser was used to see the database after it was created and to make sure the data was in the correct fields.

4. Experimental Design and Testing

The creation of an effective experiment is crucial to proving the success of our application. In order to carry out such an experiment, we had first determined what functional requirements must be implemented into our system. The requirements articulate what core features are needed by FedEx, and their drivers on a daily basis. Our experiment would then put these features to the test, demonstrating whether or not our application meets the needs of the drivers.

4. 1 Setup/Layout:

The needs of FedEx and their drivers, shaped the objectives of our experiment. *1. The application must incorporate some sort of authentication that's scalable*. Where thousands of drivers could theoretically login to view their route for the day. *2. The application must properly load stop data pertaining to a drivers route*. In this requirement, raw data must be loaded, parsed, and categorized, for later access. *3. The application must apply some sort of heuristic algorithm to the loaded stops, and therefore, create an efficient route for the driver to follow.* With these three requirements known, the system and layout that was needed became more clear.

With the knowledge that the application would theoretically operate on the android devices provided to the drivers by FedEx, it was quickly determined that our team would be utilizing android studio to create our application. Additionally, android studio offers AVDs, or android virtual devices, where our team was able to test the system on an android phone as it was being built. In terms of the layout, it was decided, a login page would be built. On this page the driver would enter his/her FedEx credentials to access their route data for the data. Additionally, we looked to store these credentials in a database to improve scalability. SQLite was determined to be the database of choice, after some testing with XAMP and PHP. After proper authentication the driver would be redirected to the routing layer of the application. In this activity layer, their stops for the day are to be ordered and presented to them in an easy to view manner. How our algorithm functioned served as the core of our project, and implementing one that was consistently effective at creating a more efficient route was imperative to the success of our system.

4.2 Datasets:

Two datasets were to be implemented into our application. Dataset one, included the login credentials for FedEx drivers(i.e. DriverID and Password). These credentials are provided to the drivers upon employment, and are stored in the backend database of the application. Dataset two consists of all stop data associated with the drivers' route on a given day. Specifically, this includes the address, zip code, any notes, delivery status, and latitude/longitude. Dataset two is stored locally in the SD card of the drivers' device, and is uploaded to the card apon authentication.

4.3 Testing:

As mentioned previously, the core of our build consisted of implementing an algorithm that consistently creates an optimal route for the driver to follow from one stop to the next. In order to test the effectiveness of an implemented algorithm, it was imperative that an initial baseline was determined prior to the route creation. A distance calculation served as an important function, to compare and contrast the efficiency of the algorithm. The distance calculation would be run on all stops in the random initial order, and then ran again on the stops in sorted order to determine how effective the algorithm served at optimizing the drivers path. Once properly implemented, the algorithm that utilized simulated annealing, visually provided a clean path with very few intersects, but further testing was crucial. When altered, different variables within this algorithm produced entirely different paths/results. These variables are temperature and coolingRate, where temperature and coolingRate together determine how long the algorithm is run. The algorithm worked in the following way: ("best" route will be the route with the least traveling distance)

1. Make an initial route using the unordered stops. This is the current "best" route.
2. Make a swap of edges between two sets of two stops.
3. Ask if the total distance traveled is less than it was.
    a. If the distance traveled is less, accept the new order of the stops as the "best" route.
    b. If the route is worse use an acceptance probability function to determine if the new worse route should be taken. The reason for this step is because there may be cases where a worst route will actually lead to a better route. The acceptance probability uses the following equation.
        i. $e^{((bestDistance - newDistamce) / temperature)}$
    c. This equation used the best route distance, the new route distance and the temperature. The temperature is the main comparison used. When the temperature is high this tells the system it's ok to take a worse off route because there is a lot of time (temperature remaining) to fix the mistake if needed. We wouldn't want to take worse routes when there is less time remaining because it wouldn't correct itself in time. If that number that gets made from the acceptance probability was greater than .99999 (this number is basically the chance that a worse off solution would be chosen) then the new worse off route would be chosen.
4. The temperature would then be decreased by the cooling rate and the algorithm would then go through the next iteration until the temperature reaches 0.

Overtime the swaps would create routes that are slowly getting shorter and shorter, in the end leaving the " best" route as the shortest and as the solution.

In testing, these values for temperature and coolingRate were altered, and affected how the algorithm behaved. Given a dataset that remained consistent throughout all testing, our "best sort" algorithm involved a very low cooling rate of 0.000001 and high temperature of 1,000,000 resulting in a path distance 2.2 times shorter than the initial with minimal overlap in routing (figure 4.1). However, it takes 9.75 minutes to complete, proving to be less than ideal. Following this test, we conducted a trial that involved a high cooling rate of .000005 with the same temperature of 1,000,000 (figure 4.2). With a higher cooling rate, more is deducted from the temperature each iteration, and the trial was over quickly in 0.112 minutes resulting in a path with multiple overlaps with a final distance only 2.05 times less than the initial. In our final trial, we attempted to get minimum to no overlaps in less runtime by utilizing the same temperature with a cooling rate of .00001. We split our initial dataset in half, and the algorithm concluded in .549 minutes with a final distance 2.3 times less than the initial proportionally (figure 4.3). It's crucial to note that a difference of 2.2 to 2.0 is pretty significant since these distances are in terms of latitude and longitude values. Markers are made with precision as much as .00001.

Trial 1

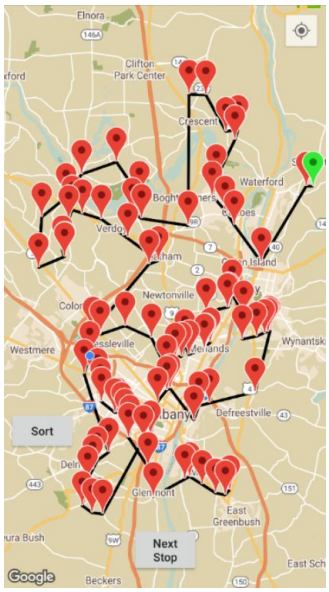

Figure 4.1

Trial 2

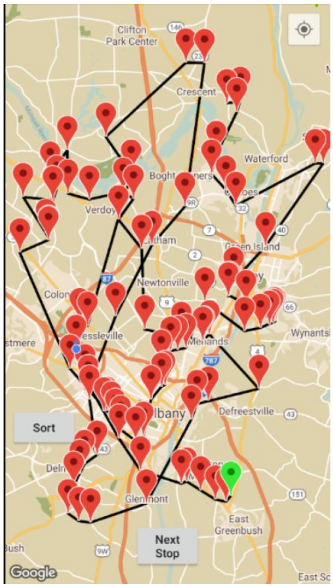

Figure 4.2

Trial 3

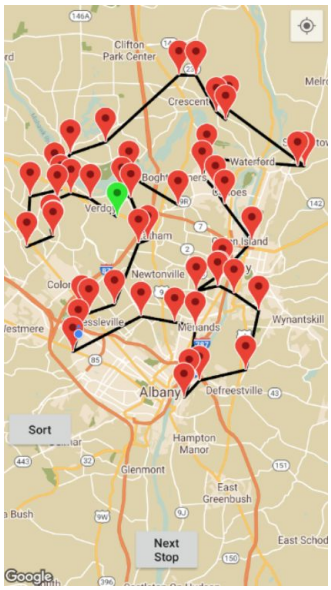

Figure 4.3

4.4 Additional testing development:

In addition to the development of our primary routing algorithm and application, attempts were made to both create a secondary, potentially competing algorithm, utilizing Google Maps API. Although this API was not used, it was a valuable asset to make comparisons against. The Google Maps API, at least the algorithm and functions that were used, provided an increase in about 30% efficiency. This is not to say the simulated annealing algorithm was better or more

efficient than the Google API. There may have been errors in the implementation and it was difficult to make comparisons since the algorithms worked very differently. For example the simulated annealing algorithm exported the results in terms of latitude and longitude where the Google implementation was in meters. The Google API implementation also did not allow much in terms of changing variables.

5. Ethics and Legal Practices

Currently, there are a limited number of ethical and legal issues faced with the construction of the application. The primary ethical issue is solely contained within end user privacy. As potential customers of such an application, we personally feel that it would be unethical to not secure the data contained within the database, solely so that customers feel their data isn't being sold, stolen or just generally misused.

Although this data is limited,mainly to just both the sender and receiver names, locations, and addresses, it would still feel inappropriate to simply release that information into the world. Additionally, as we don't know what would be left in the notes area, or what could be placed there in the future, we believe that it would also be best to be kept firmly secured. As such, some sort of encryption will likely be suggested for any data kept on both the central database, and on the actual physical phones, in the event that they are lost, stolen, or otherwise mis-placed, in order to prevent this potentially sensitive information from getting out into the public.

6. Effort Sharing

The following percentages are based on the end result code. There was research and code that was not included, either because we didn't end up needing it or we ended up changing something

Joint Tasks -
Tasks that we all contributed relatively equally

Task	Thomas	Charles	Joel
Reports	33%	33%	33%
Presentations	33%	33%	33%
Research	40%	30%	30%
Debugging	33%	33%	33%
Application Design*	60%	20%	20%

Individual Tasks -
Tasks where a majority of the work was one person but others slightly contributed

Task	Thomas	Charles	Joel
Sorting Algorithm Implementation	95%	0%	5%
Database Design and Implementation	0%	100%	0%
Reading File to Parse	10%	0%	90%
Look and Feel	10%	85%	5%

*Application design was the overall design of the application. This includes the classes used, how we should use them, data structures, how and when to use the databases, helper functions in the code, etc. The sponsor told us what she wanted and it was up to us to come up with how to do it from scratch.

7. Conclusion and Future Work

In the end, the application that was produced performed optimally given the functional and non-functional requirements that we strived to deliver on. That being said, that doesn't mean our solution came without complications. Key requirements including authentication, upload of stop data, and optimal route creation were delivered upon. However, given time limitations, full CRUD functionality and priority packages for specific stops in the route were not implemented. Ultimately, the core of our system revolved around an algorithm that consistently delivered an efficient route given a dataset of stops associated to a driver. The application was intended to improve the quality of life of every driver at FedEx, by updating and revisioning the old school system that is currently in place. Our application delivers a solution to a problem that we strived to solve. Additionally, Thomas Bridges looks to continue his work on the application in the near future.

Appendix A: Resources

Sqlite Guide -
https://developer.android.com/training/data-storage/sqlite

Sqlite Guide -
https://www.tutorialspoint.com/android/android_sqlite_database.htm

Login Structure Guide -
https://www.tutorialspoint.com/android/android_login_screen.htm

Second Traveling Salesman Algorithm -
https://www.geeksforgeeks.org/travelling-salesman-problem-implementation-using-backtracking/

Simulated Annealing -
http://www.theprojectspot.com/tutorial-post/simulated-annealing-algorithm-for-beginners/6

Google Maps API / SDK -
https://developers.google.com/maps/documentation/android-sdk/overview

Android Development Documentation -
https://developer.android.com/docs

Many Questions answered on Stack Overflow -
https://stackoverflow.com/

Wikipedia for more in depth explanation -
https://en.wikipedia.org/wiki/Simulated_annealing