TransitBuilder: A Gamified Web Application for Planning Mass Transit

Bobby Kearns
University of Puget Sound
2280 Wheelock Student Ctr.
Tacoma, Washington
bkearns@pugetsound.edu

Michael Lim
University of Puget Sound
2455 Wheelock Student Ctr.
Tacoma, Washington
mlim@pugetsound.edu

Nathan Pastor
University of Puget Sound
3683 Wheelock Student Ctr.
Tacoma, Washington
npastor@pugetsound.edu

Ian Saad
University of Puget Sound
4853 Wheelock Student Ctr.
Tacoma, Washington
isaad@pugetsound.edu

ABSTRACT

This paper describes and evaluates TransitBuilder, a gamified pedagogical tool for teaching users about the resources required to operate a robust mass transit network in an urban environment. The project is implemented as a web application to maximize accessibility, and consists of a Node.js server and a Leaflet-based client interface. This project digitizes and generalizes the transportation pedagogy of Jarrett Walker, by putting users in the shoes of a transportation planner.

1. INTRODUCTION

Frequent, reliable urban mass transit systems are more important now than ever. Over the last decade, the urban population in America has grown faster than the suburban population for the first time since the 1920's [5]. At the same time, mass transit ridership is increasing nationwide. In 2013, Americans took 10.65 billion mass transit trips, the most since 1956. And over the past 20 years ridership has increased by 37%, outpacing a 23% growth in private automobile usage [9]. Therefore, demographic changes and ridership trends indicate that mass transit is becoming an increasingly important component of the American transportation portfolio.

Unfortunately, a lack of political will has in some cases hampered needed investments in mass transit. A recent ballot initiative in King County, Washington was defeated, inducing 17% service cuts for King County Metro. A similar scenario occurred in neighboring Pierce County the year before.

This project aims to educate users about the difficulties of transportation planning, as well as the resources required to operate a robust system. Through a fun, interactive web application, users can build a transit network by drawing routes on a map of any American city. Feedback is provided by running a set of simulated trips through the user generated transit network, creating a large set of ridership data. To the best of our knowledge, there is no other application that brings a realistic yet user-friendly transportation simulation tool to the general public.

2. BACKGROUND AND RELATED WORK

TransitBuilder aims to provide a user friendly, gamified interface for transportation planning while realistically modeling the quality of the user-generated transportation networks. Therefore the project joins three categories: transportation games, ridership modeling, and transportation pedagogy.

2.1 Transportation Games

The most well-known game relative of TransitBuilder is Maxis' SimCity [6]. Now in its fifth iteration, the SimCity franchise situates users as the mayor of a fictitious town. The user must build out the municipal services of the town and zone for development while ensuring a balanced budget. The route-laying interface in TransitBuilder was inspired by that of SimCity. Users first draw the route onto the map, and then place stations or stops on top.

TransitBuilder was aesthetically inspired by Dinosaur Polo Club's Mini Metro [4]. Mini Metro situates the user as a subway planner, who must avoid station overcrowding by strategically building out a very limited number of subway lines.

These games were influential in informing the aesthetics and interface of TransitBuilder. However, the goal of these games is different from ours. Whereas game developers aim to create an entertaining, immersive world, our primary goal is realism. Although we do hope that users will find TransitBuilder enjoyable, that is a secondary concern.

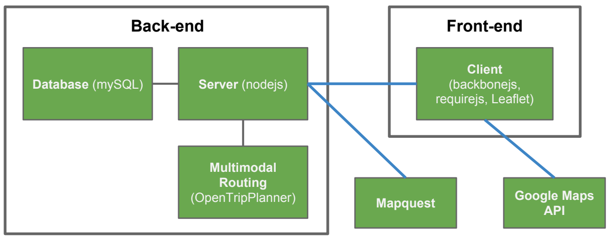

Back-end
Database (mySQL)
Server (nodejs)
Multimodal Routing (OpenTripPlanner)
Front-end
Client (backbonejs, requirejs, Leaflet)
Mapquest
Google Maps API

Figure 1: ***TransitBuilder component diagram***

2.2 Service Planning and Ridership Forecasts

Ridership forecasting is used in planning most types of transportation infrastructure, including highways, ports and mass transit. These forecasts help determine the requirements of a project, the volume of traffic it will need to accommodate, and provide a cost-benefit analysis. Ridership levels have traditionally been estimated with a four-step approach [15], outlined below:

1. A set of origin and destination points is generated based on a plethora of data including land use policy, zoning and socio-economic factors.

2. Pairs of these endpoints are linked together to form trips.

3. Transit modes (e.g walk, automobile, transit) are assigned to each trip.

4. Each trip is resolved to a particular route. For example, transit trips are assigned transit lines, and automobile routes are assigned driving directions.

Due to the complex and data-intensive nature of the four-step approach, transit agencies sometimes rely on simpler methods. These methods include deferring to the subjective judgment of professionals, or estimating ridership based on rules of thumb or comparisons to similar extant routes.

TransitBuilder's ridership model, detailed in *Section 3.1.1* uses a simplified version of the four-step approach.

2.3 Transportation Planning Pedagogy

Although Americans are often asked to vote on transportation issues, there are very few sources where they can accessibly learn about transportation planning. TransitBuilder was inspired by one of the only sources of such information; the pedagogy of transportation consultant Jarrett Walker [16]. He believes it's best to teach people about transportation planning by putting them directly in the shoes of a transportation planner. Participants in Walker's workshops are presented with a map of a fictitious service area, covered by a clear layer of plastic. The participants are given varying lengths of colored tape, representing differing transportation modes, and it's their job to plan transit in that service area by applying the tape to the plastic layer. This way, they learn about how transit agencies must serve a wide variety of demand while managing a limited budget. TranstiBuilder is essentially a digitized and generalized version of Walker's game, with the added benefit of ridership feedback.

3. IMPLEMENTATION

To achieve our goal of an accessible, fun pedagogical tool, we decided it would be best to deliver TransitBuilder over the web. Our application targets desktop computers, as we thought it would be hard to display the transit networks and associated statistics with the limited screen real estate provided by tablets and mobile devices.

TransitBuilder was implemented over the course of a semester to fulfill the Capstone in Computer Science at University of Puget Sound. Tasks were delegated among group members so as to encourage each member to master one or several components. Below we outline those components.

3.1 Back-end

The TransitBuilder back-end consists of a Node.js server and several subsidiary services including a MySQL database and an instance of OpenTripPlanner (OTP). Node.js [11] might be considered a fashionable server platform these days, and was chosen to allow us to code in JavaScript across the stack. As depicted in *Figure 2.3*, all client communication goes through the Node.js server, which is responsible for interacting with the database and OpenTripPlanner.

3.1.1 Ridership Simulation

The ridership simulation, running on the Node.js server, is the core of TransitBuilder. It is responsible for generating a realistic set of riders for each city with which we can evaluate the quality of a user-generated transit network. The ridership generation procedure consists of a simplified version of the four-step approach described in Section 2.2. We call it the two-step approach, and it is adapted from a recent DRT model [13].

1. This step combines the first two steps of the four-step approach. To determine the desirability of different portions of a city, we rely on employment and population data at the census tract level. All trips originate in a census tract weighted according to its population level, and terminate in a census tract weighted according to its employment level. In this way, tracts that are more dense will contain more trips starting, ending, or entirely contained within them. Once these tracts are selected for a particular trip, random points within each of those tracts are generated. These are the precise origin and destination points. In addition, to avoid trivially short trips, we ensure that the origin and destination points of a trip are a minimum of seven minutes apart by car. TransitBuilder currently generates 20,000 trips for each city.

2. This is essentially the final step of the four-step approach. We skip the third step because we're only concerned with mass transit. Once all the trips have been generated, they are assigned particular routes. This is determined by routing the trips through the user generated transit network, as described in the following sections.

This trip generation procedure presupposes that the population and employment levels are known for each census tract in the city, as well as the geographic outline so that random points can be generated within the tract. This data was compiled from multiple Census Bureau products. Population by census tract came from the 2010 American Community Survey, employment by census tract from the Census Transportation Planning Products (CTPP), and geographic data from Tiger Shapefiles.

The trips are saved to the database after they've been generated for a particular city. That way, the next time a user selects that city, we can skip the trip generation process entirely.

3.1.2 GTFS

To determine how the simulated trips interact with the user generated transit network, TransitBuilder represents that network as a GTFS feed [8]. GTFS (General Transit Feed Specification) is a standard pioneered by Google and Portland's TriMet, and consists of a collection of text files. The text files contain transit agency information, define each route, each trip on a route, and the time at which each trip reaches every stop. GTFS provides a standardized medium for transit agencies to present their networks and timetables to multimodal direction services such as Google Maps. TransitBuilder uses OpenTripPlanner [12], an open source multimodal routing server, to route the simulated trips through the transit network.

3.1.3 Routing

Hosting an OpenTripPlanner instance allows us to pass in start/end points along with a mode (e.g. walk, transit, car) to get the most optimal route between those points. This routing engine serves two purposes. The first purpose is used in the drawing process for bus routes as they must be confined to streets, unlike subways (compare the yellow and blue lines in *Figure 5*). As will be discussed in *Section 3.2*, users build transit routes by drawing a series of waypoints on the map. If it's a bus route, we snap it to roads by using OpenTripPlanner to get driving directions between all those waypoints.

The second, more important service that OpenTripPlanner provides is routing the simulated trips through a user's transit network. This process begins when a GTFS bundle representing a user's transit network is passed back to the server. OpenTripPlanner then compiles a graph based on street data taken from OpenStreetMap and the GTFS feed. With the graph built and loaded into memory, we can begin routing the simualted trips through it to update ridership for that session. See *Figure 2* for a visual display of the process. The resources required to build the graph cause some limitations which will be discussed in *Section 4.2.4*.

3.1.4 Database

The database consists of three main components, which allows TransitBuilder to cache and persist key data. Because finding all the census tracts in a city is such a resource intensive process, the first component is used to cache census data for each city loaded up by the user. When a city is selected by a user, the system will first check the database to see if the census tracts comprising that city have already been determined. If the city is not found, the server will initialize the process by which all census tracts for a city are extracted from the list of tracts in the state. The system

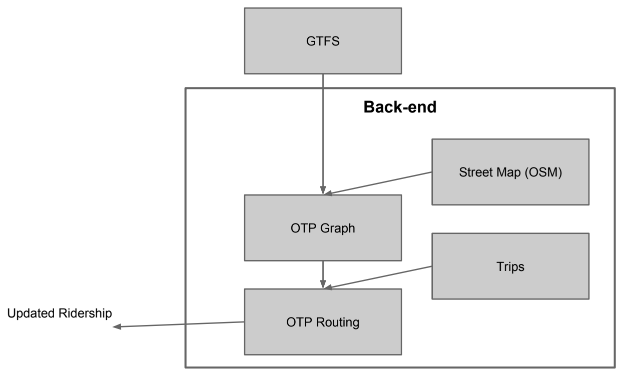


Figure 2: ***Ridership update procedure. The GTFS feed is passed from the client to the server and compiled into an OTP graph with OSM data. Ridership is then updated by routing the simulated trips through the OTP graph.***

will then cache this information in the database for future use. In this way we use a lazy loading technique.

The second component in the database stores the simulated trips for each city. This process is very similar to how the census tract information is stored. If a city has never been selected by any user then it will not have any trips generated for it. Once a city has been selected by the user, the backend will generate trips to be used for later use in the system. The main purpose for these two components were to have a quick response when a user selects a city.

The last component for the database stores user sessions. This allows the user to come back and edit their transit system at a later time to help better serve the virtual riders. When a user first visits TransitBuilder they are prompted with a choice of loading or creating a new transit system. If the user attempts to load an existing session, the system will go to the database and fetch the session if it exists and serve it up back to the user. This process also serves as a purpose to update ridership for the user's transit system. The other option creates a new session that is only saved to the database after the user clicks the simulate button. This will save all the routes and ridership generated by the user.

3.2 Front-end

The client side logic is organized with Backbonejs [2], a framework which helps provide a nice separation of data and user interface logic by encouraging an MV* pattern. Data is represented in Backbone Model objects, while Backbone Views are responsible for rendering that data to the DOM and capturing user input. We also used Requirejs [14] to modularize our JavaScript code. Unlike programming languages such as Java, C and PHP, vanilla JavaScript has no mechanism for importing other scripts. Requirejs adds this functionality.

To allow users to build their transit networks in an intuitive way, as well as to provide an effective means for visualizing demographic information, TransitBuilder uses a "slippy map," the type of standard, drag-to-pan map used by services such as Google Maps and OpenStreetMap. This functionality is achieved with the help of the open source Leaflet library [10]. Leaflet is an industry standard library which simplifies the process of deploying a production grade map. It manages the task of fetching and displaying the image tiles which make up the base map, handling user input to draw transit lines, as well as overlaying our supplied demographics.

To draw custom layers on top of the Leaflet map, we make use of the GeoJSON [7] specification. This format allows for basic geometric structures such as points, but also more complicated elements like lines and polygons. This gives us a standard way with which to define all of the geographic features we display such as census tracts, transit routes, and route stops. GeoJSON also allows us to have a consistent way to represent our data across our application. Once it is created, we can simply pass the GeoJSON object wherever it is needed, for example, giving it to Leaflet to display or passing it back to the server to save.

To build routes, users begin by drawing a line directly on the map. When the drawing is complete, they are presented with a pop-up. Here they give the route a name, specify the mode, and specify the service frequency. After entering this information, they must add stops to the route to provide entry and exit points for the passengers. The current version of TransitBuilder only supports bus and subway planning.

To provide feedback to the users on the quality and cost of their transit network, several statistics are displayed to the user. These are visible in the right panel in *Figure 3*. The statistics are as follows:

- **Total Demand Satisfied**. This is the average percentage that each simulated trip spends riding mass transit. For example, a 20 minute trip that includes a 4 minute subway ride, 6 minute bus ride and 10 minutes of walking would be considered 50% satisfied.
- **System Cost per Day**. This is calculated by multiplying the total revenue hours per day for each route by an operational cost for each route's mode. Operational costs by mode are derived from a comprehensive set of transportation statistics released by the APTA [1]. A bus costs $129.42 per revenue hour, and a subway costs $209.31 per revenue hour. (Revenue hours refer to the hours a transit line is in operation. For example, a route that takes one hour to complete and has 10 buses serving it would entail a total of 10 revenue hours)
- **Passengers per Revenue Hour**. This statistic represents the efficiency of each route, and is calculated by dividing the number of simulated passengers that ride the route by the number of revenue hours.

TransitBuilder also makes use of the popular Bootstrap [3] framework. This gives it a familiar look and feel, while also providing a more satisfying user experience and more robust cross platform support.

4. EVALUATION

With a project of this scope, unforeseen difficulties are expected. Below we discuss TransitBuilder's primary use case, as well as some areas for improvement.

4.1 Primary Use Case: New Session

A user accessing TransitBuilder is first presented with the option of starting or restoring a city session. After picking the former option and selecting an America city, the map pans to that city. At this point, the user can toggle the employment and population density layers from the right panel to aide in network planning. The user can also begin to build their transit network, by drawing lines on the map and placing stops on top of them. As the stops are added, the operational cost of the system is updated based on the total revenue hours. When the user has built out the transit network sufficiently, she can press the stoplight button in the bottom right corner. This begins the process described in *Section 3.1.3*, whereby the GTFS bundle is sent back to the server, an OpenTripPlanner graph of the city is built, and ridership is updated by routing the simulated trips through the graph. This process takes several minutes, depending on the size of the city, after which the updated data is saved to the server database under the session name. To view the updated ridership the user must reload the city session. At this point, statistics including total demand satisfied and route efficiency are displayed in the right panel. With this new information, the user can now modify the transit network by adding and deleting routes.

4.2 Areas for Improvement

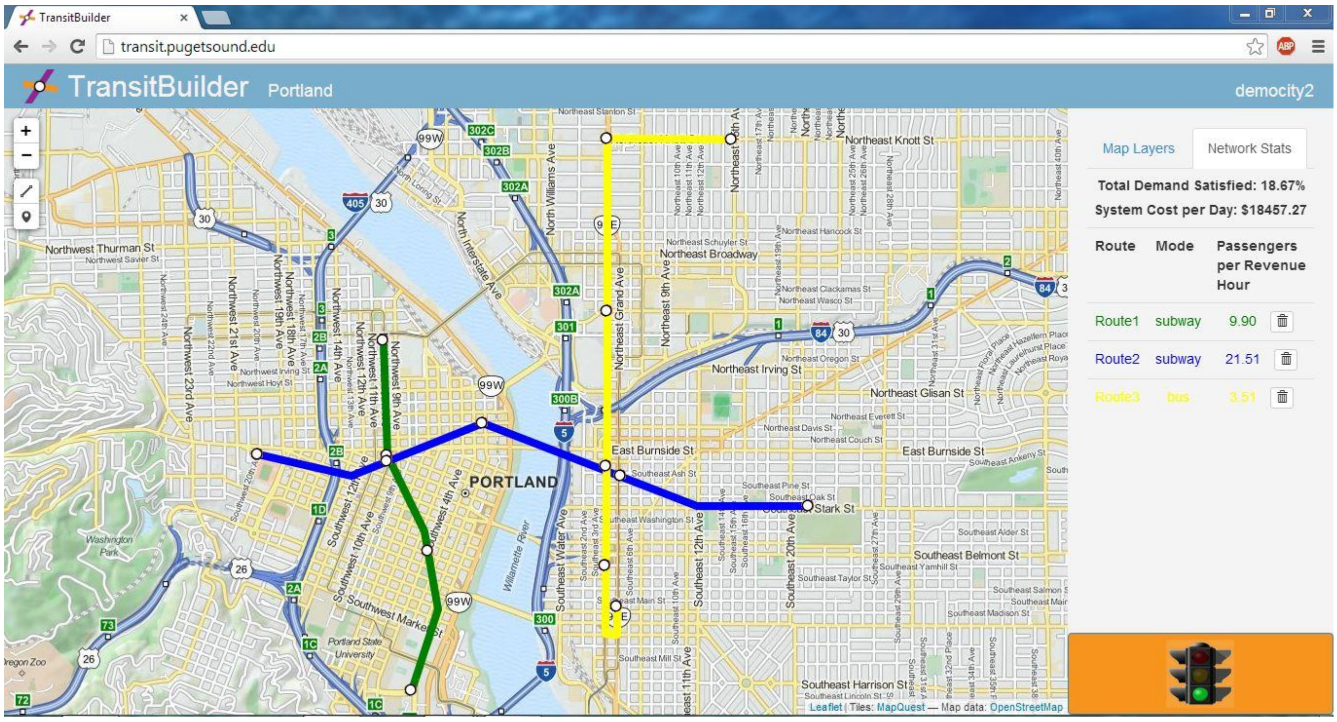
4.2.1 Simulation Robustness

One of the biggest improvements to the simulation that can be done is the type of trips that are generated. Currently there is just one type of trip being modeled. By only anchoring origins in tracts weighted according to population and anchoring destinations in tracts weighted according to employment, all of our simulated trips are essentially commuter trips. To model other types of trips, the system might generate a portion of trips solely based off of population needs, a portion of trips based off employment needs, as well as the trips that are a mix of population and employment needs.By varying these factors, we'd create a simulation the that more acurately reflects real ridership patterns.

Another improvement that would add a more realistic feature to TransitBuilder would be trips departing at different times throughout the day rather than just all departing at one time. Currently the system is routing all trips at 9:30 AM, with this improvement of having different departure times the system will be able to show peek ridership times.

Finally, the last improvement that would help give TransitBuilder simulation a more realistic and robust feel would be generating the amount of trips based off population rather than having a set number. The plan for this would be to calculate the population for the given city and than generate a value based off that city's population. Currently TransitBuilder is generating 20,000 trips regardless of the city. For large cities like New York City and Los Angeles, 20,000 is not sufficient to accurately model ridership.

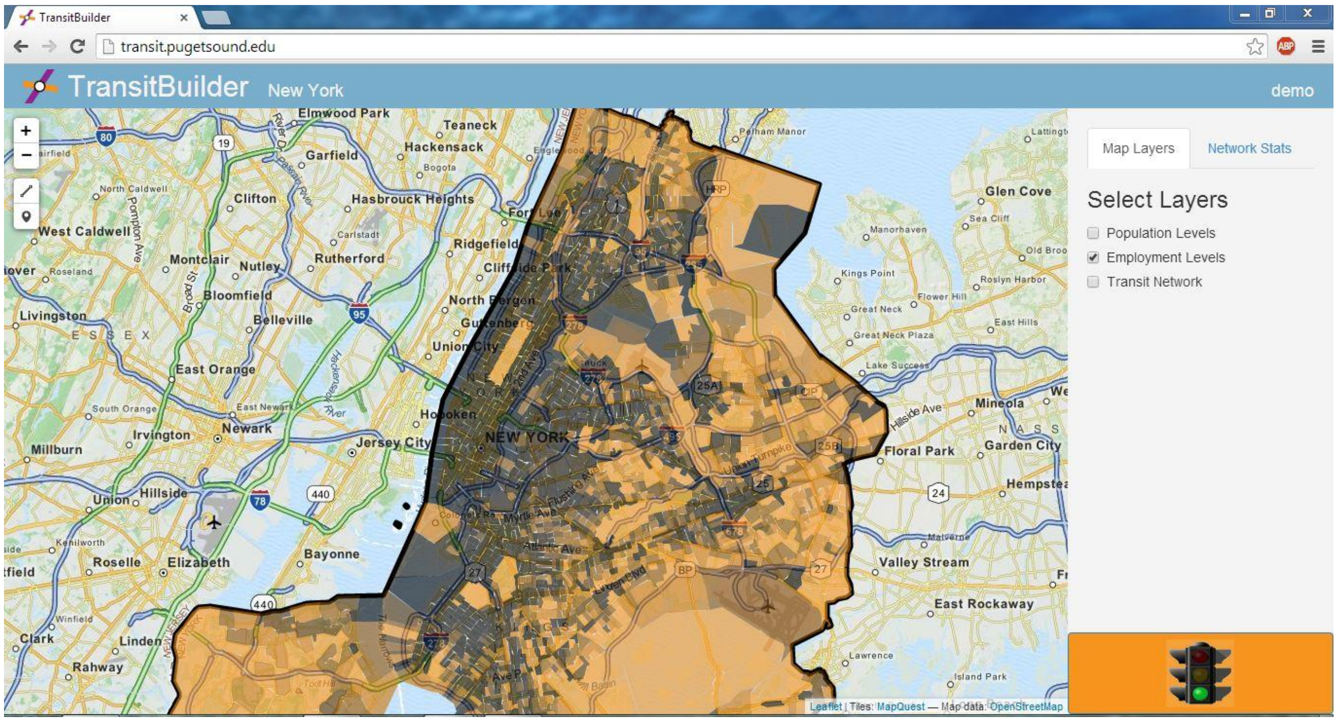
The improvements listed above are all concerned with trip generation. There is also room for improvement in determining how those trips interact with the transit network. Our initial plan for the simulation split trips into *dependent* and *discretionary* categories. The former category is completely dependent on mass transit. Therefore, no matter how poor



Total Demand Satisfied: 18.67%
System Cost per Day: $18457.27

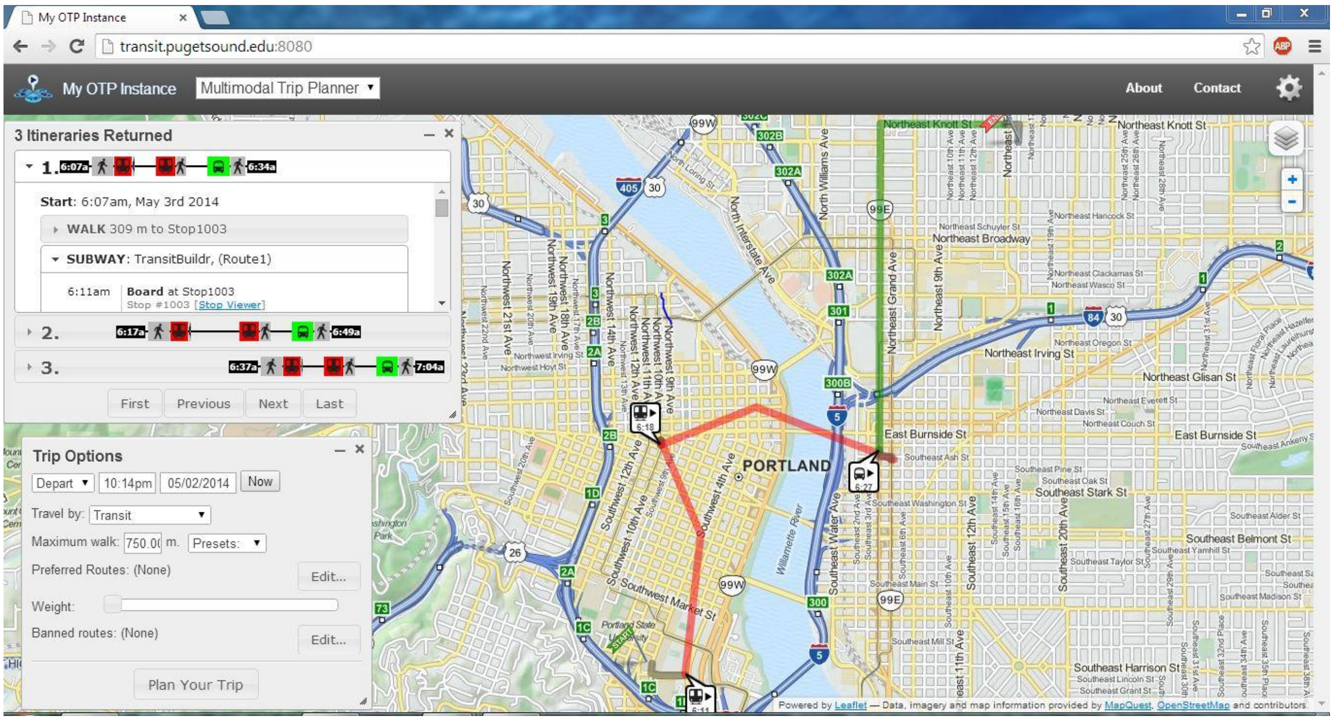
Route	Mode	Passengers per Revenue Hour
Route1	subway	9.90
Route2	subway	21.51
Route3	bus	3.51

Figure 3: *The TransitBuider user interface, showing two subway routes (green and blue) and a bus route (yellow). Users can draw routes by selecting the line tool in the upper left corner of the map, and dragging the mouse across the map. Route stops are specified by selecting the marker tool and clicking on the route lines. Statistical feedback on the quality of this transit network is displayed in the right panel.*



Select Layers
- [ ] Population Levels
- [✓] Employment Levels
- [ ] Transit Network

Figure 4: *The TransitBuider user interface, showing employment densities by census tract. Darker tracts contain higher employment densities. Map layers can be toggled from the right panel.*


My OTP Instance
transit.pugetsound.edu:8080
My OTP Instance
Multimodal Trip Planner
About
Contact
PORTLAND
Powered by Leaflet — Data, imagery and map information provided by MapQuest, OpenStreetMap and contributors.

3 Itineraries Returned

1. 6:07a … 6:34a

Start: 6:07am, May 3rd 2014

WALK 309 m to Stop1003

SUBWAY: TransitBuildr, (Route1)

6:11am Board at Stop1003
Stop #1003 [Stop Viewer]

2. 6:17a … 6:49a

3. 6:37a … 7:04a

First Previous Next Last

Trip Options

Depart 10:14pm 05/02/2014 Now

Travel by: Transit

Maximum walk: 750.00 m. Presets:

Preferred Routes: (None) Edit...

Weight:

Banned routes: (None) Edit...

Plan Your Trip

Figure 5: *The OpenTripPlanner client, showing a trip's route through the transit network depicted in Figure 3*

the transit service is, they would ride. But the latter category only rides mass transit out of convenience. So if their trip required too much walking, for example, we would consider their entire trip unsatisfied. These distinctions would amplify the benefits of designing a good transit system. Not only would a good transit system better serve the *dependent* riders, it would also attract new *discretionary* riders if it is sufficiently convenient.

4.2.2 Budget

While TransitBuilder in its current state is a fairly realistic simulation and modeling tool, one of the biggest inaccuracies is the lack of a budget constraint. Although TransitBuilder does calculate and display the overall operational cost of the designed transit networks, it doess not utilized or restrict expenditures in any way. Adding a starting budget, perhaps based on the selected city's population, and then being limited to the construction of a transit network that fits within that budget would make TransitBuilder even more realistic.

A further improvement that would add to the realism would be the inclusion of feedback between ridership and revenue. Allowing ridership of the transit lines to generate revenue, which would then be put back into the users budget, would allow for a more dynamic simulation, where a user would go through many cycles of building and testing their network. The inclusion of fare generation and dynamic budgeting would also serve as another way of evaluating a user's network. A well designed and utilized transit system would provide a surplus, allowing for additions and modifications to the network, while a poorly made system that was largely unused would result in a deficit, and ultimately lead to failure.

4.2.3 Ridership Analytics

As discussed in *Section 3.1.3*, by routing simulated trips with OpenTripPlanner, the TransitBuilder simulation creates a robust set of ridership data. This data includes walking time, walking distance, transit distance, boarding and alighting stops, and much more for each simulated trip. This is emphasized in *Figure 5*, which depicts a rider route through a transit network using the OpenTripPlanner client. Therefore, although the TransitBuilder interface only provides feedback at the route and system level, we have the data to be much more precise. If we had more time to build out the application, we would increase the resolution of those ridership and cost statistics, for example by showing which route segments are underperforming or which stops are overcrowded.

4.2.4 Server Scalability

One of the most difficult obstacles in moving forward with this application is graph creation. As mentioned in *Section 3.1.3*, OpenTripPlanner compiles a graph with a GTFS feed and street data. This process takes up a large amount of memory, to the extent that our server was unable to compile Texas or California. The graph must also be compiled every time the transit network is updated because the GTFS feed is altered. Currently this is not much of an issue, merely an inconvenience of a few minutes for the user as the graph compiles. However, as the user base increases, compiling many graphs quickly becomes a very large workload. A large increase in our server quality and quantity would need to occur before we released.

5. DISCUSSION

Our main goal for this project was to implement a user-friendly, accessible, gamified tool for allowing users to plan transit networks in any American city. We believe that mass transit is more important than ever, and wanted to help educate citizens who are often asked to vote on transportation issues. The TransitBuilder prototype that we have implemented is largely successful in achieving this goal.

TransitBuilder was largely inspired by the pedagogy of Jarrett Walker, and it provides a more accessible means of conveying similar information. Walker's transportation planning course can cost upwards of $400 per participant, which makes it inaccessible to the vast majority of citizens. However, because TransitBuilder is a free to use web application, is has the potential to be accessible to anyone with a computer and an internet connection. Furthermore, Walker himself is essential for providing feedback on the transportation networks that participants of his seminars design. With our ridership simulation, we have effectively automated that feedback.

One of the most interesting outcomes of this project is a personal one. Perhaps the most difficult challenge we faced was constructing, following, and adapting a timeline for this project. While we all had experience working on sizeable group projects, the scope and scale of this application made planning and estimating time a much more difficult task. In the beginning, when we tried to lay out our initial plan, we found it almost impossible to estimate how long each portion of the project would take. Because of the required research, testing and integration, a seemingly small hurdle in one section could drastically impact the progress in many other areas. Therefore we tried to leave our schedule as flexible as possible, overestimating time when we were unsure. However, even with these safety features built in, we still faced some unexpected setbacks, with corresponding issues in our timeline. For example, we expected to be able to access employment, demographic and geographic data for a city with a simple, generalized API call. The actual process proved to be much more complex, and this affected other areas of the application. Without this data, we could not accurately generate trips, and without trips, we were unable to evaluate the user's transit network. Pitfalls like these proved to be a valuable lesson in the importance of timelines and project management.

6. FUTURE WORK

If we continued developing TransitBuilder, there is one large problem we would need to address. As discussed in *Section 4.2.4*, running our ridership simulation requires an immense quantity of computing power. Our server, which has 8GB of memory, is sufficient for serving several concurrent sessions of TransitBuilder. However if we wanted to scale up the project and release it to the public, we would have to expect hundreds if not thousands of concurrent sessions. With each session requiring upwards of 0.4GB of memory (for OTP graphs), we would need access to much more computing power. We of course don't have the monetary resources to accommodate that. In light of this major limitation, we would need to rethink the entire ridership simulation. Instead of running our agent-based simulation to determine ridership levels on the transit networks, we'd likely have to rely on heuristics and estimates. Making this change would have the added benefit of reducing the delay that users currently experience in updating ridership, as the OpenTripPlanner graph compilation and ridership update procedure takes several minutes to complete. Alternatively, we could implement our own OTP graph that wouldn't require GTFS bundles to be compiled in. That way, we could have one street graph of the United States loaded into memory, and GTFS feeds from each user session could be passed to the OTP routing engine dynamically. Following this path would require us to familiarize ourselves more with the OTP code base.

If we were able to solve the above problem, there are several other features we would need to implement to get TransitBuilder production worthy. The first is a budgeting system, as described in *Section 4.2.2*. By imposing a budget, we would do better in conveying some of the tradeoffs inherent in transportation planning. For example, users might have to decide between building a subway line downtown, or building out bus lines to the less dense residential areas.

We would also like to add user authentication, which was beyond the scope of our timeline. We were more interested in implementing the ridership simulation and the route planning, which is really the core of the project. Because of this, the current version of TransitBuilder allows anyone to load a city session, update it, and save it back to the server. It would be better to authenticate users as they enter the site, and then associate city sessions with those users. Once we have authenticated users, we would be able to add a social media aspect. At the very least, we would allow a mechanism for users to share the transit networks they built. This would also gamify the application more, by creating an implicit competition for who can build the best transit network for the least amount of resources in a particular city. There could even be leaderboards where people upload their network's GTFS feeds and other users could download, run, and edit them to explore what can be improved upon.

Lastly, to bolster the primary objective of TransitBuilder we would like to add the ability for users to compare their transit networks to the cities' actual transit systems. Now that most real transit agencies provide GTFS feeds to the public, this could be accomplished by running our simulated trips through both the user generated GTFS feed and the real GTFS feed of the city. Comparing the cost and ridership satisfaction of the user's created system to that of existing networks would be very beneficial.

7. CONCLUSION

TransitBuilder largely accomplished the goals that we set out to achieve. It provides a user-friendly interface for planning transit in any American city, and utilizes a realistic ridership model to determine the quality of user-generated transit networks. In this way, it builds on the work of Jarrett Walker by providing similar pedagogy with a much more generalized and accessible platform.

The project was also a valuable learning experience for us. Prior to beginning the implementation, we had very little experience with web development and JavaScript. This project has introduced us to many commonly used JavaScript frameworks and libraries. We gained valuable insight into the intricacies of the JavaScript language, and the differences between it and Java. In particular, we learned to implement the MV* pattern using BackboneJS, server-side scripting using Node.js, and asynchronous module loading using Requirejs. We also dealt with the issues of incorporating and modifying plugin features with standard libraries. For instance, much of our drawing functionality was adapted from various Leaflet plugins, and integrating those features with Leaflet and BackboneJS was a significant learning experience. Lastly, the instance of OTP that we run required several feature modifications from the released version. In order to make these changes we had to dive into a massive codebase, figure out what it was doing, and find the portions that we needed to change.

8. ACKNOWLEDGEMENTS

We would like to thank Professor Brad Richards for his help over the course of this project, and Paul Monaghan for setting us up with a server - and a domain name - to host the TransitBuilder back-end. Thanks are also in order for the many developers behind the open-source software used in this project, including Leaflet, OpenTripPlanner and Node.js.

9. REFERENCES

[1] American Public Transportation Association. 2011. 2011 NTD Tables: Urbanized Area Data. Retrieved 5/11/14 from http://www.apta.com/resources/statistics/Pages/NTDDataTables2011.aspx

[2] Backbonejs. Retrieved 5/11/14 from http://backbonejs.org/

[3] Bootstrap. 2014. A Frontend Framework for Developing Responsive Projects on the Web. Retrieved 5/13/4 from http://getbootstrap.com/

[4] Dinosaur Polo Club. 2014. Mini Metro. Retrieved 5/9/14 from http://dinopoloclub.com/minimetro/

[5] Dougherty, Conor and Robbie Whelan. 2012. Cities Outpace Suburbs in Growth. Retrieved 5/9/14 from http://online.wsj.com/news/articles/SB10001424052702304830704577493032619987956

[6] Electronic Arts. 2014. SimCity. Retrieved 5/9/14 from http://www.simcity.com/

[7] GeoJSON. 2014. A Format for Encoding Geographic Data Structures. Retrieved 5/13/4 from http://geojson.org/

[8] Google. 2012. What is GTFS? Retrieved 5/9/14 from https://developers.google.com/transit/gtfs/

[9] Hurdle, Josh. 2014. Use of Public Transit in U.S. Reaches Highest Level Since 1956, Advocates Report. Retrieved 5/8/14 from http://www.nytimes.com/2014/03/10/us/use-of-public-transit-in-us-reaches-highest-level-since-1956-advocates-report.html?_r=0

[10] Leaflet. 2014. An Open-Source JavaScript Library for Mobile-Friendly Interactive Maps. Retrieved 5/11/4 from http://leafletjs.com/

[11] Node.js. 2014. node.js. Retrieved 5/13/14 from http://nodejs.org/

[12] OpenTripPlanner. 2014. Multimodal trip planning and analysis. Retrieved 5/9/14 from http://www.opentripplanner.org/

[13] Pastor, Nathan. 2014. The Feasibility of Citywide Public DRT: Door-to-door Bus Service in Tacoma.

[14] Requirejs. 2014. A JavaScript Module Loader. Retrieved 5/11/14 from http://requirejs.org/

[15] Transportation Cooperative Research Program. 2006. Fixed-Route Transit Ridership Forecasting and Service Planning Methods. Retrieved 5/9/14 from http://www.tcrponline.org/PDFDocuments/tsyn66.pdf

[16] Walker, Jarrett. 2011. What Makes a Good Planning Game? Retrieved 5/9/14 from http://www.humantransit.org/2011/08/what-makes-a-good-planning-game.html