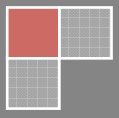
2009

Connecting People and Events: Multi-Modal Routing and Dynamic User-Generated Content

Honors Thesis

Project to develop a multi-modal routing algorithm for The Ohio State University's campus and greater Columbus area, and to provide a platform to encourage students to participate in diverse activities and events on the campus.

Prabhjyot Chawla
The Ohio State University
5/11/2009



HONORS THESIS

Connecting People and Events: Multi-Modal Routing and Dynamic User-Generated Content

Presented in Partial Fulfillment of the Requirements for the Degree Bachelor of Science with Distinction in Computer Science and Engineering of The Ohio State University

By

Prabhjyotsingh Ramindersingh Chawla

The Ohio State University

2009

Thesis Committee:

Dr Paul A.G. Sivilotti, Advisor

Dr Rajiv Ramnath

Abstract

The Ohio State University is one of the largest campuses in the United States. New students can have an overwhelming experience to adjust to a new place, a new educational system, new surroundings, and also make new friends. At the same time, The Ohio State University brings together students with diverse backgrounds, and has more than 900 student organizations on campus. Each student organization relies on its own means to communicate their events and activities with students on campus. For this reason, it is difficult for student organizations to find students who are interested in their activities and to promote events to large group of students. In addition, it is a challenge to travel around the city without a car. There are ample online tools to provide information such as schedule for public transportation, building locations, student activities but they have not been connected together in a useful fashion. There is a need to help students feel more comfortable in their initial days at the University, help them get around Columbus during their time in college, and to help them sort through the overload of information to find interesting activities on campus. This research project aims to develop an application that helps students find their way from one location on campus to another, encourages them to explore the diversity on the campus, reach out to their peers, support those who proactively seek opportunities for growth, and hopefully contribute to the campus in a positive manner.

The thesis explores the need for such an application, presents a domain analysis of the project and defines requirements. One of the central ideas of the application is to encourage the community to develop data for the site. Individuals can directly add events, activities, information of student organizations on the site. Additionally, the thesis develops a multi-modal routing algorithm for public

transportation. The Dijkstra's algorithm was used as the shortest path algorithm for the project. Three important modifications were made to the graph in order to apply Dijkstra's algorithm for routing over public transportation. Firstly, the vertex of the graph used for bus routing requires latitude and longitude of the bus stop, along with the time when the bus arrives at the bus stop. Hence, the graph used in bus routing has three dimensions – latitude, longitude and time. Secondly, three different kinds of edges were added to the graph – bus, waiting and walking edges. Lastly, the concept of 'weight' of an edge in Dijkstra's graph was modified from it being equal to the distance between the two nodes, to 'effort' required from the part of the passenger. 'Effort' is calculated based on the amount of time travelled, number of transfers taken, and distance walked in the journey.

Acknowledgements

I would like to thank the following:

1. Dr Rajiv Ramnath for being an important resource of technology, for networking opportunities, and for including the development of the mobile application as a capstone course in his class.
2. Dr Rephael Wenger for providing insight on testing Dijkstra's shortest path algorithm.
3. College of Engineering, the Leggett Family Fund and The Ohio State University Student Government for supporting the project with academic scholarships.
4. My web development team, which included Chad Sowald, Pete Koelsch and Anh Nguyen, who helped research student organizations, get data from different stakeholders, develop a web-site and evolve the concept of the project.
5. My mobile application development team, which includes Markus Rogosinsky, Shinta Salim, Sheetal Ghadse and Ho Mak.

Finally, and most importantly, I would like to thank Dr Paul Sivilotti for his guidance, encouragement and indispensable mentoring over the course of the entire year.

Table of Contents

Table of Figures

List of Tables

CHAPTER 1

Introduction

The Ohio State University has one of the largest student campuses in the United States. The University had 61,568 students enrolled in Autumn Quarter 2008 (Statistical Summary, 2009). The university attracts students from many diverse backgrounds and countries. In the academic year 2007-08 over 3,600 students from 104 countries came to study at the university (Office of International Affairs, 2007-08). Moreover, there are about 900 student organizations on campus and thousands of activities and events are organized on campus every year (Activities & Organizations, 2009). The University has a large campus, which is spread over 15,910 acres, and has 911 buildings on campus (Statistical Summary, 2009). For a new student, the first few months at the university can be an overwhelming experience. There is an overload of information and it becomes a challenge to recognize activities of interest.

There is a need to help students feel more comfortable in their initial days at the University, help them get around Columbus during their time in college, and to help them sort through the overload of information to find interesting activities on campus. In particular, it is difficult to get around Columbus without a car. Schedules of bus services can be confusing, and online web routing services may not be easy to use. This research project aims to develop an

application that helps students find their way from one location on campus to another, encourages them to explore the diversity on the campus, reach out to their peers, support those who proactively seek opportunities for growth, and hopefully contribute to the campus in a positive manner.

One of the central ideas of the application is to encourage the community to develop data for the site. Individuals can directly add events, activities, information of student organizations on the site. Essentially, the idea is to extend the concept of wikis to the realm of Geographic Information Systems (GIS), wherein individuals can contribute to the web application by helping keep the data of public transit systems up to date. Others could add information about buildings on campus directly onto the map, or warn others about road constructions and road blockages in their neighborhoods.

The application is an experiment in developing a much larger application where digital map content can be generated in the form of a wiki. India, China, and many countries in Africa and Eastern Europe lack digital map content. Generating and maintaining digital map content is an expensive and labor intensive activity. After the successes of applications like Wikipedia, it is but natural to experiment with the possibility of implementing GIS wikis. New road constructions could be updated within days instead of waiting for digital maps to be changed, which can sometimes take as long as 6 months to a year. Citizens could come together and suggest new bus routes to the public transportation agency. Business travelers from America could feel at home in India because their smart phones would be able to display up-to-date maps of cities in India that have been modified by conscientious locals. Bus schedules for remote places in Africa could be added to the website by users, and online routing services for the public transportation could be extended to those areas.

However, as interesting and challenging these possibilities are, the project scope had to be narrowed. The scope was narrowed down to developing a routing algorithm for public transportation for the campus and greater Columbus area. Dijkstra's shortest path algorithm is a classic single-source shortest-path algorithm (Cormen, 2003). The vertices in a directed graph are connected by non-negative edges. Starting from a given vertex (the source), the algorithm creates a shortest path spanning tree, i.e. it finds the shortest path between the source vertex and every other vertex in the tree that is reachable from it. It does this by always adding to the growing tree the closest vertex from the source. Dijkstra's shortest path algorithm is an example of a greedy algorithm. The running time of the algorithm with a priority queue implementation is $\Theta(n \log n + m \log n)$, where n is the number of vertices in the graph and m is the number of edges in the graph.

Dijkstra's algorithm was used as the shortest path algorithm for the project. However, certain modifications had to be made to the graph in order to accommodate for the differences between routing for buses as compared to routing between cities over a highway network. Firstly, buses run on specific schedules. So a route between two locations depends on the time of the day when the journey is made. In contrast, a shortest path between two cities is unlikely to change depending on the time of day. Secondly, a passenger making a bus trip has to occasionally switch buses to complete a single journey. The passenger might even have to cross the road and take a bus from the other side of the road. Thus, the algorithm has to accommodate for the possibility of a passenger having to wait at a bus stop, having to walk to a bus stop which was close by, and having to get off one bus and take another. Lastly, a passenger prefers to make a trip that is convenient. A passenger usually prefers a trip that has the least walking, least amount of waiting time at bus stops, no bus transfers (switching from one bus to

another), and least amount of time to complete the journey. The total amount of distance traveled is usually less important to the passenger taking a bus than it to someone driving in a car. However, sometimes walking a little extra or taking a bus transfer (switching from one bus to another) might save the passenger a lot of time in the total journey.

In order to apply the Dijkstra's algorithm for multi-modal routing of public transportation the following changes were made to the graph. Firstly, a vertex in a graph representing the highway network of the US requires latitude and longitude to be described; whereas the vertex in the graph used for bus routing requires latitude and longitude of the bus stop, along with the time when the bus arrives at the bus stop. Hence, the graph used in bus routing has three dimensions – latitude, longitude and time. Secondly, in order to allow the passenger to switch buses, walk to another bus stop and wait at a particular bus stop three different kinds of edges were added to the graph – bus, waiting and walking edges. If a bus route exists between two nodes then they are connected by a bus edge. Waiting edges connect two nodes in the graph that belonged to the same bus stop. Walking edges connect two nodes that are close enough in space-time such that the passenger can walk from one to the other. Lastly, in order to provide the most convenient route to the passenger the definition of 'weight' of an edge in Dijkstra's graph was modified from it being equal to the distance between the two nodes, to 'effort' required from the part of the passenger. This *new* weight is calculated based on the amount of time travelled, number of transfers taken, and distance walked in the journey. As a direct consequence, the algorithm treats the three different edges differently. Bus edges are usually preferred over walking and waiting edges. Staying on one bus is usually preferred over switching buses. However, walking edges are taken and/or buses are switched if significant time savings in the total time of the journey is obtained.

In the thesis, the concept of an online application to connect students on campus with student organizations, events on campus, public transportation and a GIS wiki have been presented, along with detailed description and analysis of the algorithm developed to find optimal routes for public transportation. In Chapter 2, the work products for requirements of such an application, including the domain analysis, solution analysis, use cases and use case diagrams are presented. In Chapters 3-6, the database design, explanation of Dijkstra's algorithm and application of Dijkstra's algorithm are outlined. Chapter 7 talks about the future steps for the project. Finally, in Chapter 8, the discussion on the application and the analysis of the routing algorithm are summarized.

CHAPTER 2

Requirements

Domain Analysis

The need for the project grew from personal experience and observation. These can be clubbed into the following three broad categories.

- Diversity on Campus
- Getting around on Campus and in Columbus
- Digital Maps

Diversity on Campus

The first few months on campus can be an overwhelming experience for someone who is new to Columbus or to the university. The diversity of the people on campus and the sheer number of people and events on campus can make the experience difficult. There is an overload of information that it takes time for an individual to become comfortable with new environment. Some facts of OSU are below.

- The Ohio State University has the largest campus in the United States, with 53,715 students enrolled on the Columbus campus in Autumn Quarter 2008, and with a total of 61,568 students enrolled across all its campuses overall. The headcount of University employees is just under 40,000 (Statistical Summary, 2009).

- Over 3,600 students from 104 countries joined the University in the academic calendar 2007-08 (Office of International Affairs, 2007-08).
- The Ohio State University has nearly 900 student organizations on campus (Activities & Organizations, 2009). There is no central location for information of all events and activities taking place on campus. Information of events organized by the university can be found on one site, activities in Columbus on another, and information of events organized by student organizations on separate individual websites, emailing lists and/or Facebook. As a consequence, it is difficult for a student to stay well informed with events and activities on campus, and proactively seek opportunities for personal and professional growth.
- Yet, it can sometimes be difficult to make friends. Especially say, if an international student joins the university in winter quarter, and does not live in the dorms.

Getting around on Campus and in Columbus

- The University has 463 building in the Columbus campus alone, and a total of 911 buildings when the other campuses are included (Statistical Summary, 2009).
- The University covers 15,910 acres of land which includes the Columbus campus, local campuses, University airport, golf course and the Ohio Agricultural Research and Development Center (Statistical Summary, 2009).
- COTA provides public transportation services to Columbus, but accessing its website is slow, and confusing. Ability to view a bus route on a map, locate bus stops and the schedule of the bus at the bus stop are located in distinct areas of its website.
- The city does not have bicycle lanes. However, bicycles are popular on campus.
- There is no established web-service to provide routes with walking and/or bicycle paths. There is no web-service which provides a combination of multiple modes of transportation.

- It is difficult to get around the city of Columbus without a car. Moreover, driving a car can be difficult because the campus area has a number of one way lanes.

Digital Maps

- Some of the fastest growing economies in the world include Argentina, Brazil, China, Czech Republic, Hungary, India, Malaysia, Mexico, Poland, South Africa, South Korea, Taiwan, Thailand, and Turkey. Most of these countries lack digital map content. In addition, most of these countries do not have English as a preferred language of communication. Existing web routing services cannot be extended into these countries without developing digital map content (at a great price) and by making customizations specific to the culture of the area (language, syntax of addresses, etc).
- According to D. Venugopal (CEO of Advanced Space Technologies & Services based in Bangalore) "some select (Indian) cities have digital maps. Some skeleton services are available but not widely spread. The problem is the need to constantly update the data base. Lack of data on exact small street names is another reason."
- For example, Vadodara, a city in western India, with a population of over 1.6 million people does not have adequate digital map content. Figure 1 shows the map of Vadodara in Google Maps and Live Maps at approximately the same zoom level. Google Maps includes locations of some landmarks and few major roads. Information about the roads is missing, including names and type of road, e.g. whether it is a one lane, highway, etc. Live Maps is missing most of the details of the map. On the other hand, Columbus with a population of over 0.7 million people has detailed map content. Figure 2 shows Google Maps and Live Maps of Columbus, Ohio at approximately the same zoom level.

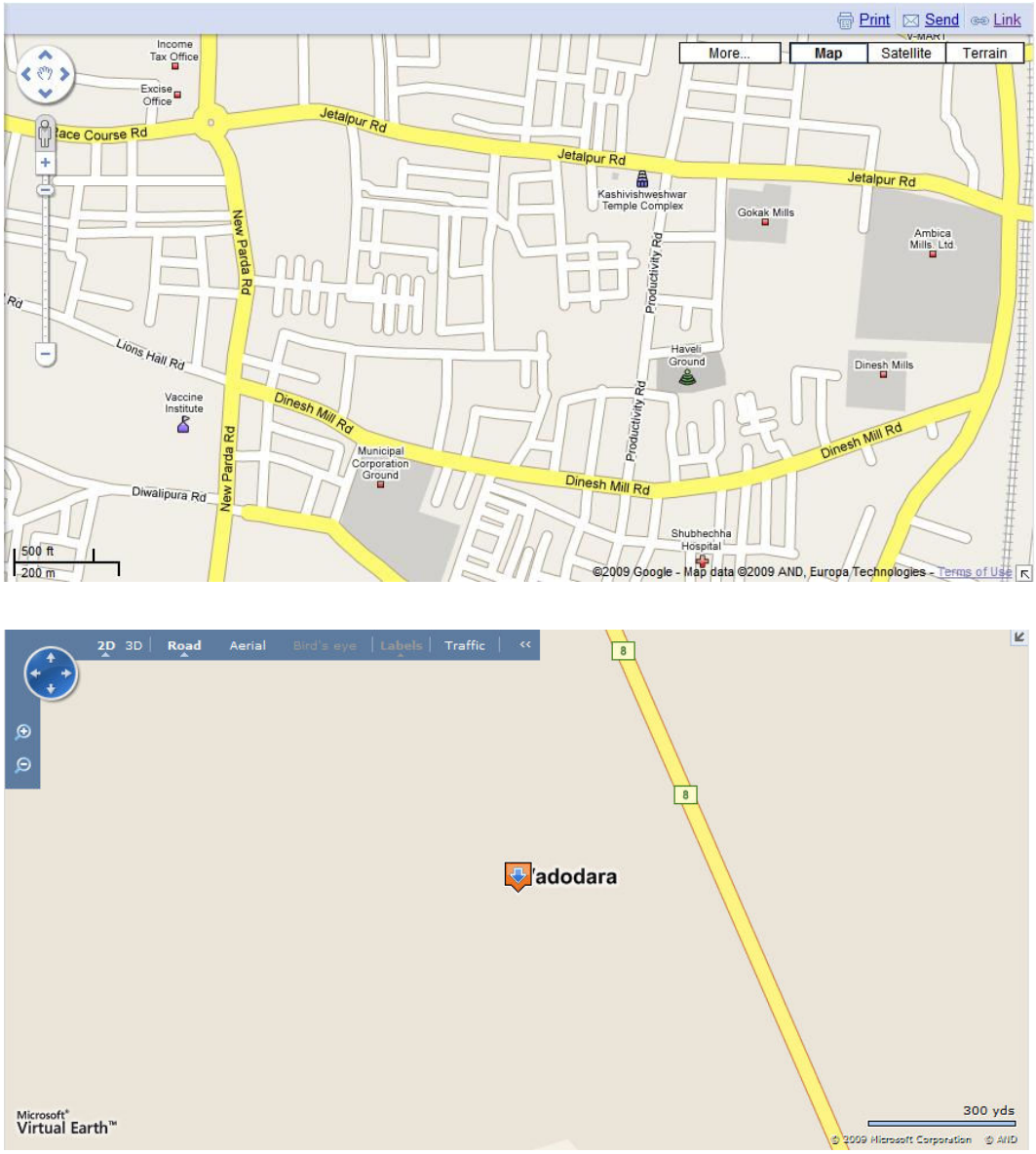
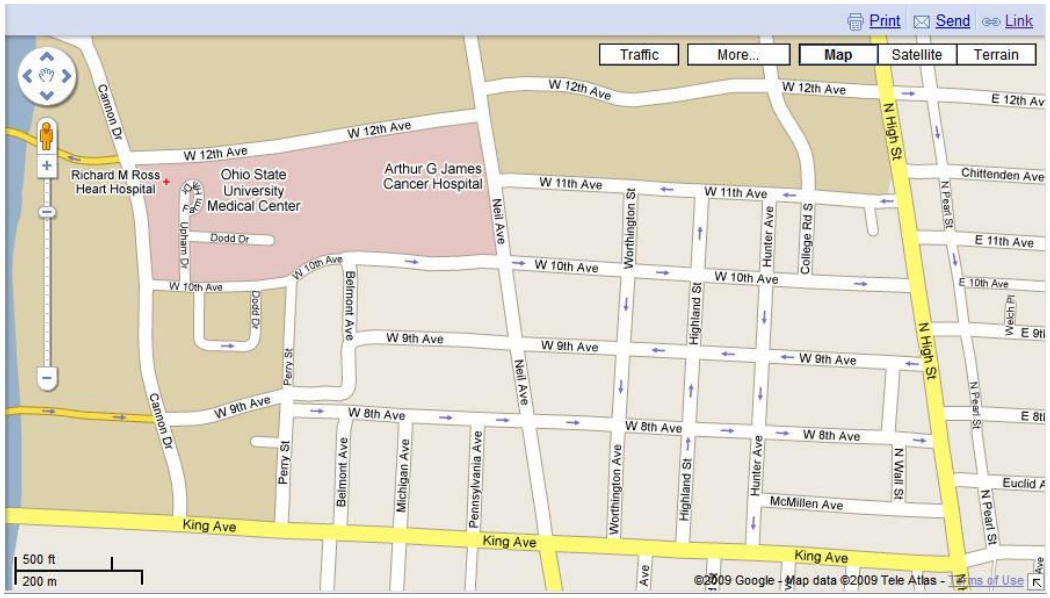


Figure 1: Google & Live map of Vadodara (Gujarat, India)



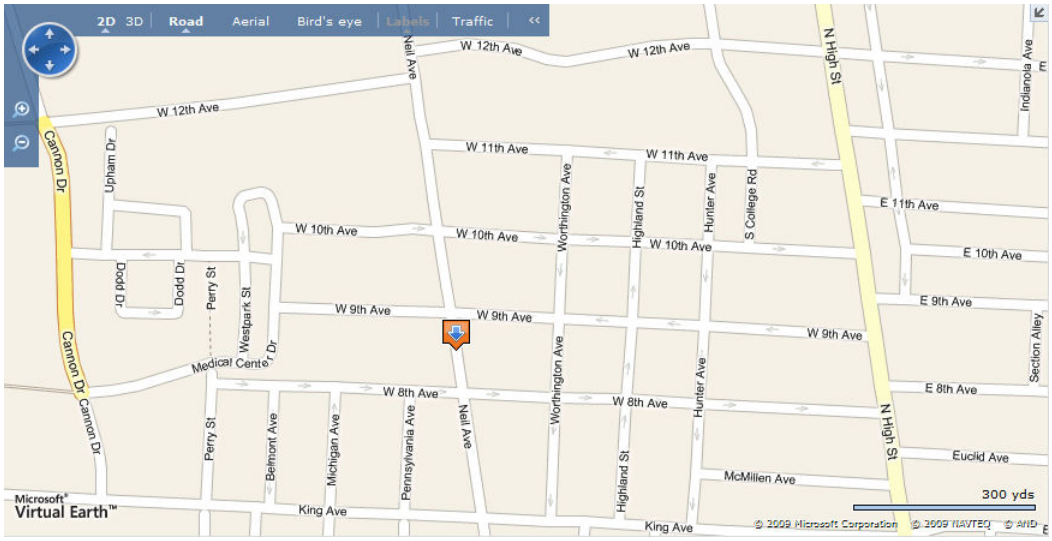


Figure 2: Google & Live map of Columbus

Survey of Student Organizations

Students volunteering for student organizations are already playing a leadership role on campus, and by narrowing down the definition of users to them helped to focus attention to solving specific needs, and hopefully providing tangible value to all students on campus.

In order to get a better understanding of how student organizations communicated with their members and promoted their events, a survey was developed and emailed to all student organizations. 48 responses were received, and their responses were analyzed. All student organizations did not respond to all questions, and in some cases they provided multiple answers. The objective of the survey was to better understand the end user and to get input on developing requirements for the application.

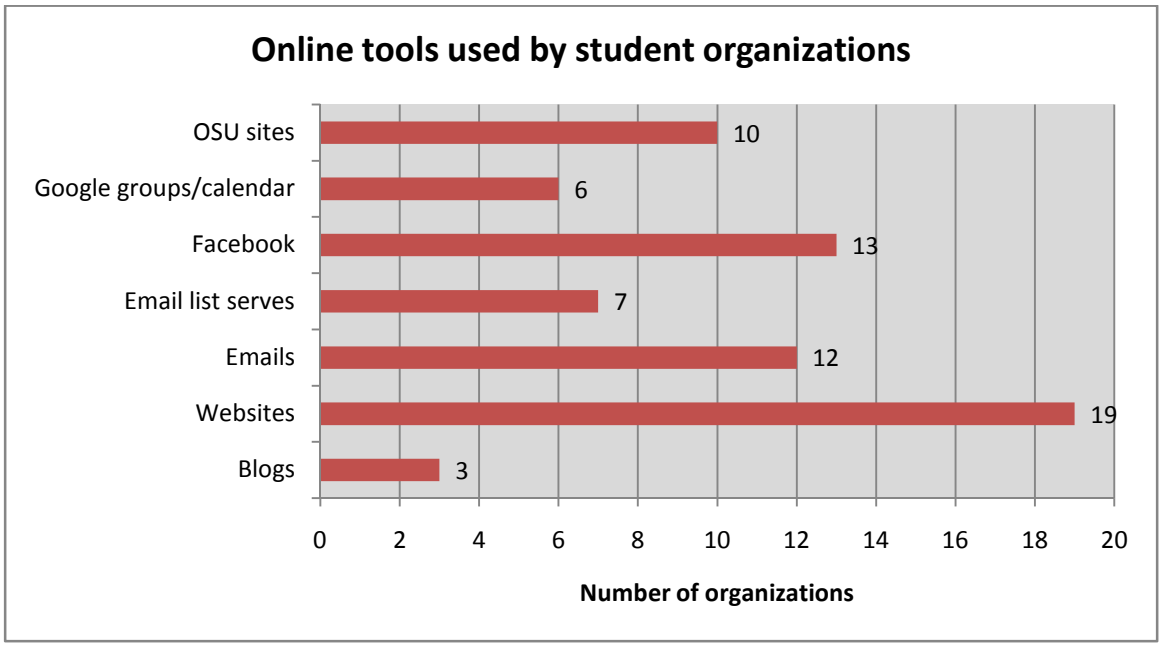
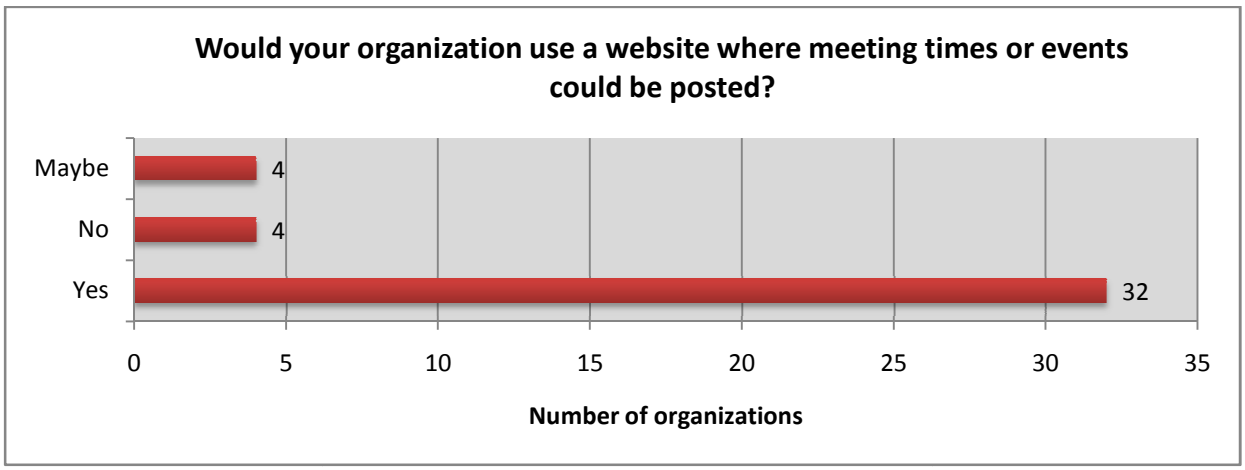
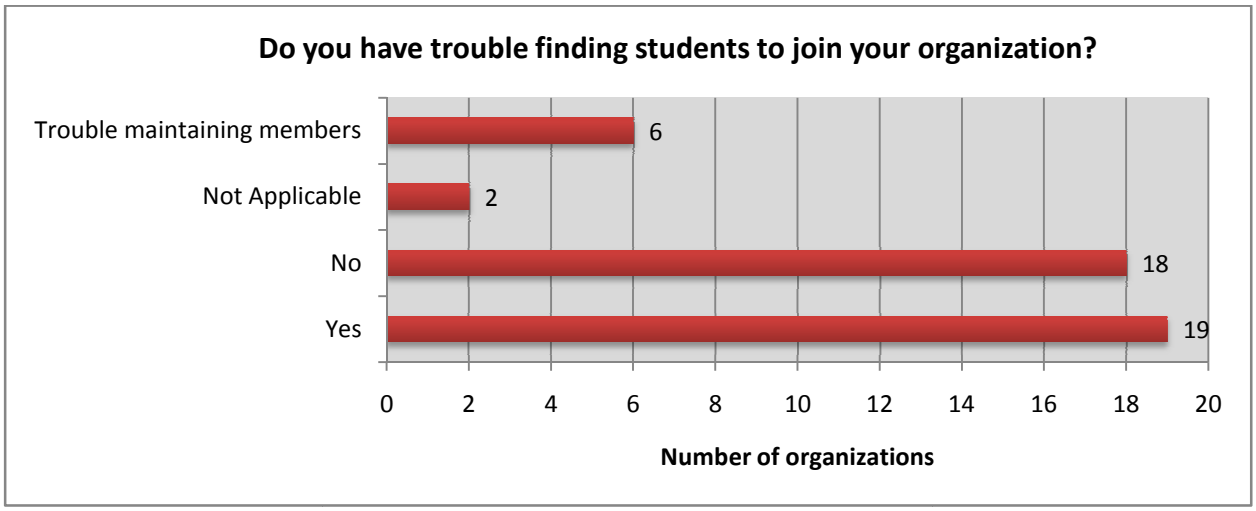
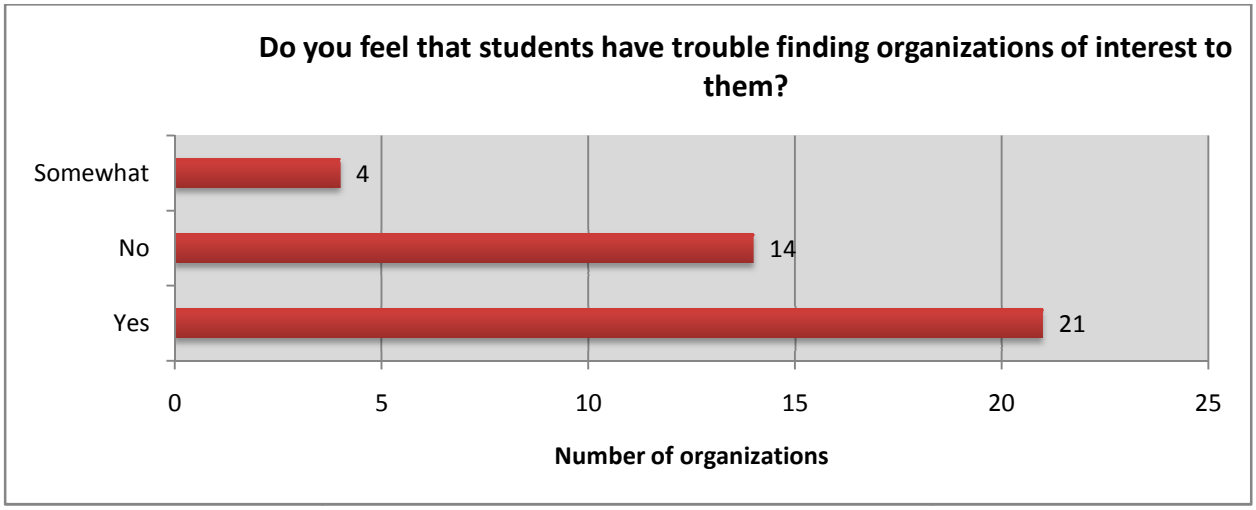


Figure 3: Responses obtained from student organizations

Figure 3 continued







Ideas proposed by student organizations for an application:

- Provide reminders to events via email.
- Provide a “wall” for people to post comments on.
- A recommendations engine for events and student organizations.
- Advertisements for events and organizations.
- General information pages on organizations.
- Categorize student organizations.
- Allow for pictures to be uploaded.

Observations from the survey:

1. Not one online tool meets all the needs for the student organizations. Often the student organizations relied on more than one tool to help them communicate.
2. Students want a lot of basic features (such as an organization profile page, wall, categorization of activities and a place to upload pictures) that can help structure communication of events and student organizations.
3. Organizations want an application that would help promote their activities and events. 32 out of 40 clearly mentioned that they would use a website that would help them promote their events. Some organizations suggested that they wanted to place advertisements about their events/organizations, and some wanted a recommendations engine for events.

Routing is the core product

There are sites such as yelp.com and zvents.com that already provide information of events in the city. However, such sites do not target college students specifically and hence do not contain events specific to the campus. If a student needs to find information about events on campus they are more likely to trust a university web-site, or the site of the student organization itself. Probability for a student to proactively search a third party application for events on campus was considered low. Therefore, in order to attract students on the site, the application has to fulfill a specific need of the students.

Based on feedback received from presenting at Europa Undergraduate Research Forum at Department of Computer Science and Engineering, discussions with students in the hallway and personal experience, it was established that people find it difficult to travel using public transportation in Columbus. The web-sites of public transportation agencies are difficult to navigate and at the point of writing the thesis there is no established web-site that routes over Columbus using Campus Area Bus Service (CABS) and Central Ohio Transit Authority (COTA). Hence, multi-modal routing over public transportation was picked as the core product of the application. Routing for public transportation seemed complex and was a high risk task, so it was important to work on it as a priority.

Helping students find routes for public transportation could fulfill an almost daily need of the students. Once the students use the site for routing, hopefully they would realize that the site also contains rich content specific to events and activities on campus, and that would encourage them to participate and take initiative in activities on campus.

Solution Analysis

Multi-Modal Routing

Develop an optimal web-routing algorithm for multiple modes of transportation for the campus and greater Columbus area. The website will give the users the ability to find optimal routes by walking, using the CABS or COTA buses, bicycles or even driving. The website will provide the information to the user in an easily accessible manner, along with the ability to customize routes.

Develop an online Community

Students will be able to browse all events taking place on campus from one location. The database would be easily searchable and the events would be categorized so that users can easily find events of interest. Categorizing events will also help student organizations their target audience directly. Individuals, organizers, and student groups will also be able to add events directly to the site.

Geographic Information Systems (GIS) Wiki

Users will have the functionality to modify/add data to the map. For example, changes in road name and new constructions can be recorded by the users directly on the map, in the form of a wiki. Users would be able to add/change route or schedule information of public transportation services. Moreover, users will be able to provide feedback on the changes made by other users. Depending on feedback received from the community, a particular user could be given greater privileges in helping maintain the integrity of map data for a particular locality. Thus users would be organized in a hierarchical manner, and misuse of the site would be discouraged.

Modifiable & Scalable

The project could be extended to countries that lack digital map data, e.g. India and African countries. A GIS wiki could be used to generate map content for these regions. Later, web routing facilities can be extended to regions with enough digital map-content. For example, a user in India would be able to add rivers, roads and buildings from his neighborhood directly onto the map and record specific information about these features. Another user in Mumbai would be able to find a route to her office which involves taking an auto rickshaw, a local train and then finally switching over to a taxi.

Use Case Diagram

Use Case Diagram for Routing of Public Transportation Systems

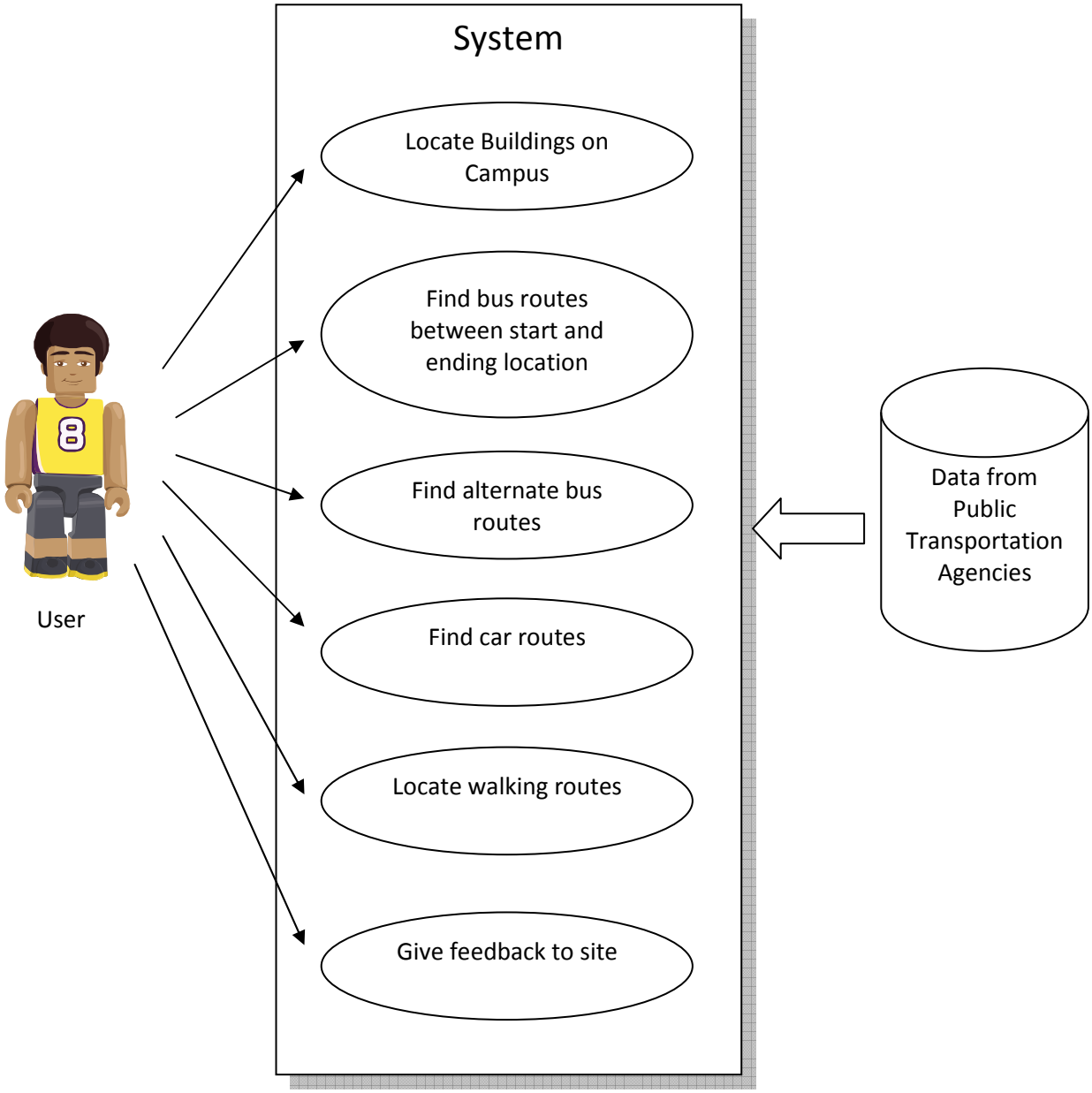


Figure 4: Use Case Diagram - Routing

Use Case Diagram for Events

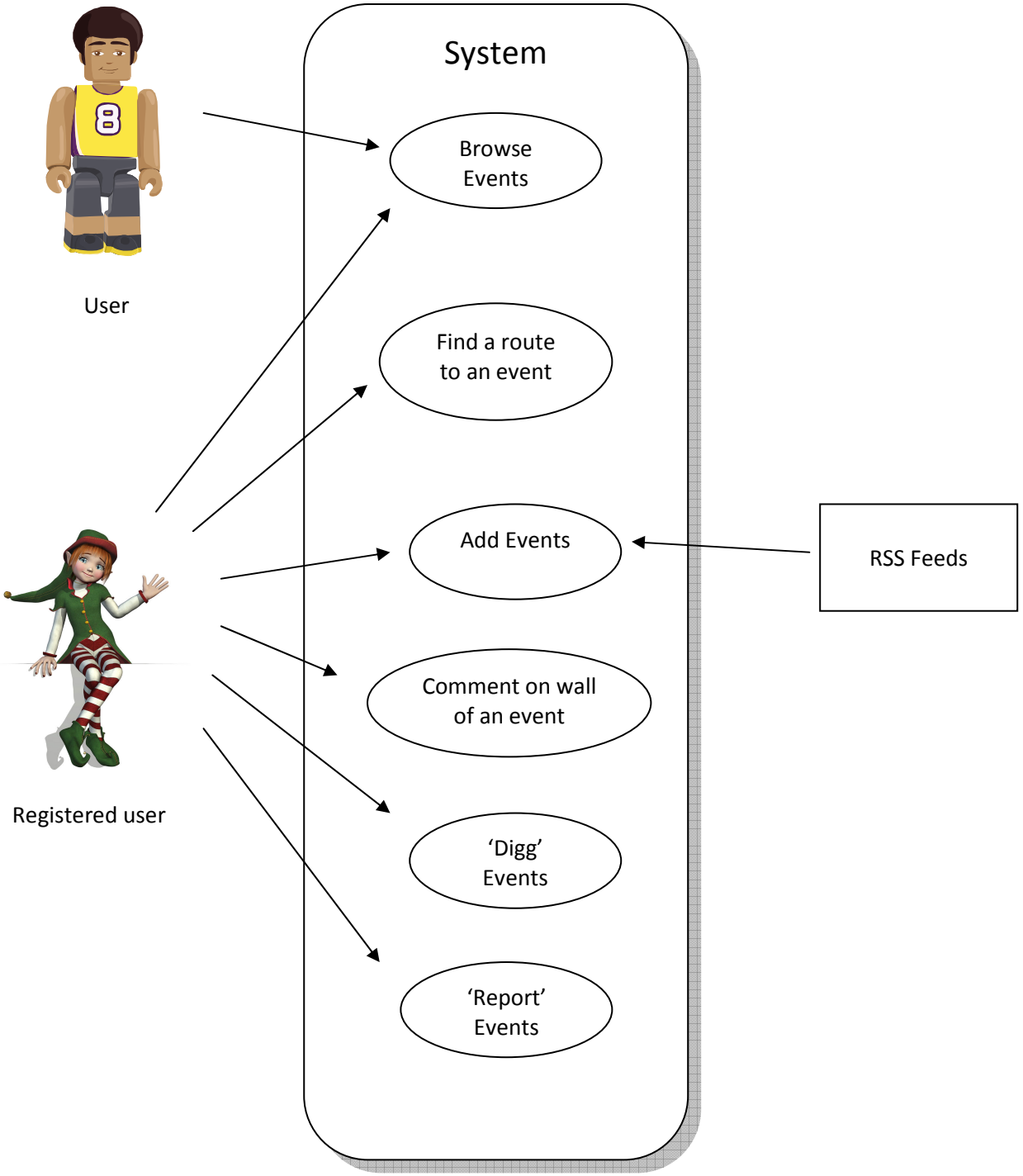


Figure 5: Use Case Diagram - Events

Use Case Diagram for Student Organizations

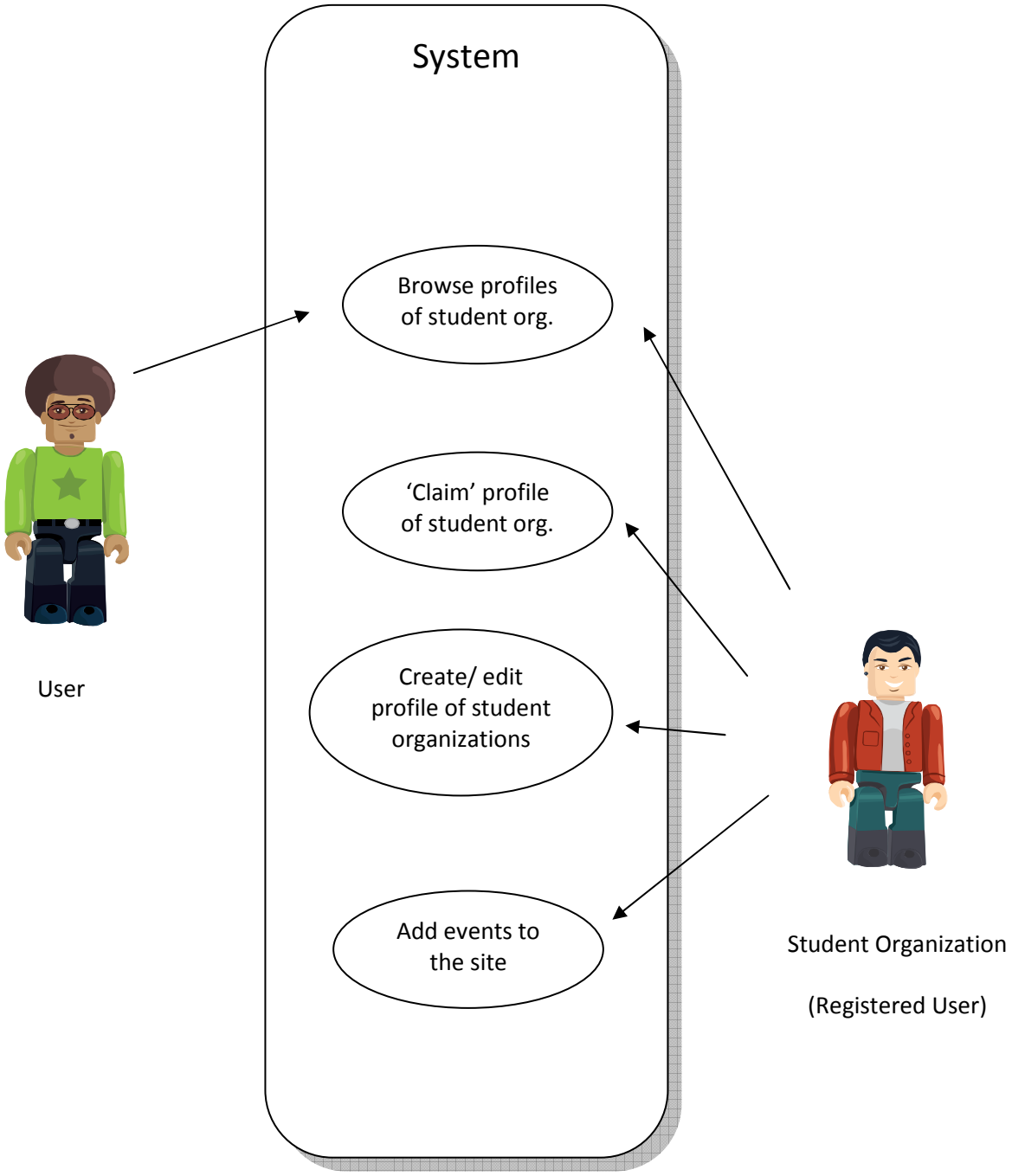


Figure 6: Use Case Diagram - Student Organizations

Use Case for Geographic Information Systems Wiki

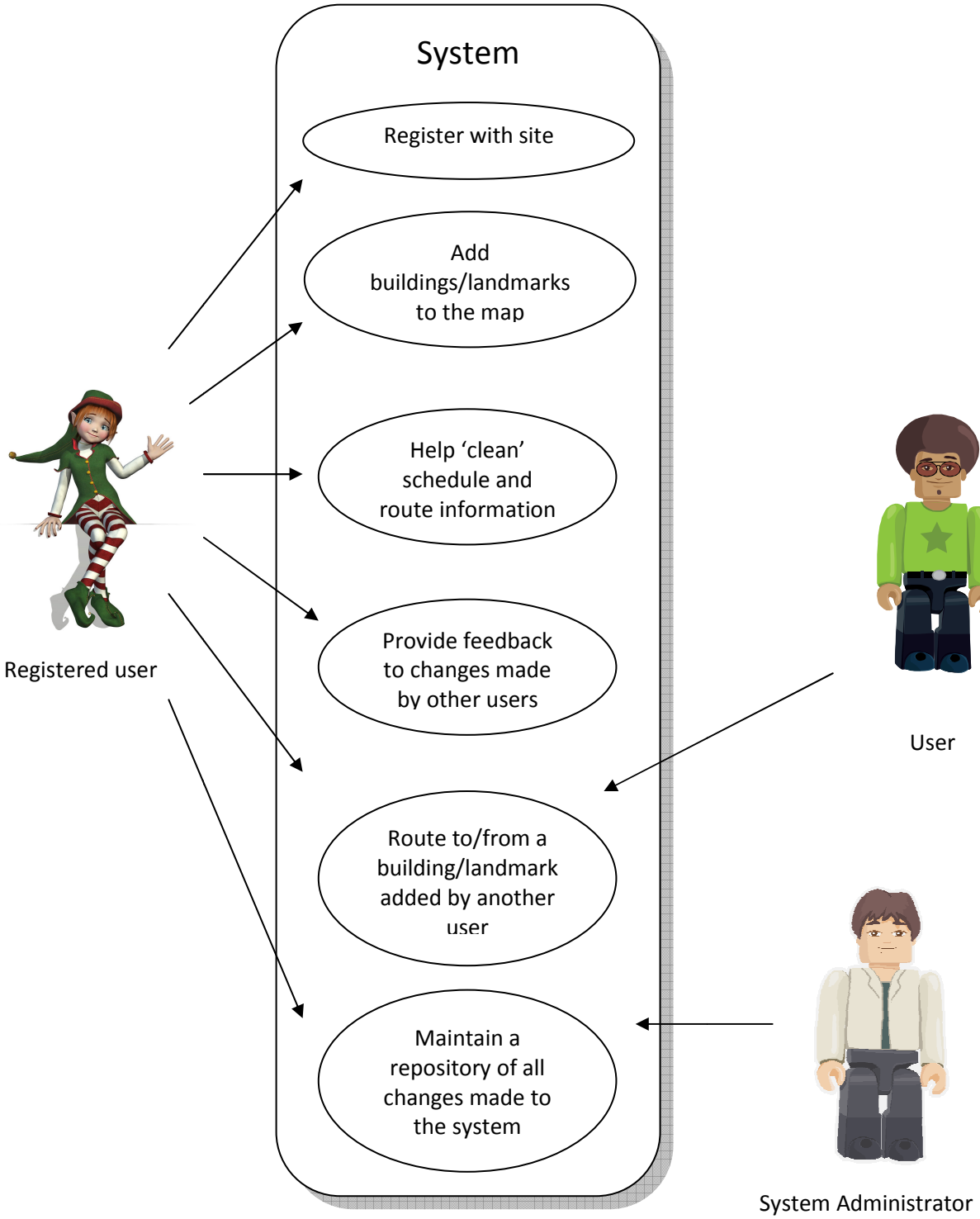


Figure 7: Use Case Diagram - GIS Wiki

Use Cases

Name: Find Bus routes

Actor: Student on campus

Normal Flow:

- User visits the site, and enters the start location and the end destination of the trip. The user also enters the preferred start time of the journey, and chooses to find a 'bus route.'
- The user can enter names of buildings on campus and those names will auto-complete. If the user chooses to enter the street address, then the user does have not to add "Columbus, OH" to the address. The site will assume that the partial street address is an address in Columbus, Ohio.
- The site generates the bus route and plots it on the map. The site plots the starting bus stop, the route of the bus and the bus stop where the user is supposed to get off the bus. If the user has to transfer onto another bus, the site will also plot the intermediate bus stop. If this transfer involves walking to another bus stop, then the site will also plot the walking route to that stop.
- The directions of the route are also displayed in English.

Exceptional Flow:

- If the user is not happy with the route, the user can leave feedback on the site.

Name: Find Alternate Bus Routes

Actor: Professor on campus

Normal Flow:

- User visits the site, and clicks on 'View bus routes.'

- The user can plot the start location and the end destination of the trip. The site will show the bus stops close to these locations.
- The user can enter names of buildings on campus and those names will auto-complete. If the user chooses to enter the street address, then the user does have not to add "Columbus, OH" to the address. The site will assume that the partial street address is an address in Columbus, Ohio.
- The user will see a list of the names of bus routes on the website. If the user checks the box next to the name of a bus route then the route is displayed on the map. The user can display multiple routes on the map at one time. The routes will be displayed in multiple colors.
- Bus stops on the bus route will also be displayed. The user can click on the bus stop and see information about the bus stop, and when the bus is scheduled to arrive the bus stop.
- This feature will help the user plan their own trip on the site by looking at various bus routes.

Name: Find Car route

Actor: Resident of Columbus

Normal Flow:

- User visits the site, and enters the start location, the end destination of the trip, and chooses to find a 'car route.'
- The user can enter names of buildings on campus and those names will auto-complete. If the user chooses to enter the street address, then the user does have not to add "Columbus, OH" to the address. The site will assume that the partial street address is an address in Columbus, Ohio.

- The site contacts Google's server, plots the route on the map, and displays directions in English.

Name: Browse Events on Campus

Actor: Student on campus

Normal Flow:

- User visits the site and clicks on 'Events'.
- The user will browse events taking place on campus. The user will be able to browse events by date and categories. The user can also search for events using keywords.
- The user clicks on an event to view details.
- If the user finds an interesting event that she might want to promote to other users, she can "digg" it.
- The user can locate the event on the map and find a route to get to the event.

Exceptional Flow:

- If the user finds the event offensive/illegal then the user can login to the system and 'report' the event to the administrators of the site.

Name: Browse student organizations on Campus

Actor: Students on campus

Normal Flow:

- The user visits the site, and clicks on 'Student organizations.'

- The user will be able to search for a student organization by keywords. In addition, the user will search a student organization by categories, e.g. type of organizations, number of members in the organization, frequency of events.
- The user can see the profile of a student organization, their pictures, and a list of the events they are going to host.
- The user can click on the name of an event and view its details.

Name: Claim 'profile' of student organization

Actor: Student organization on campus

Normal Flow:

- All student organizations that have been registered with the University would exist in the database of the site.
- An email would be sent to the President's & Vice President's of all student organizations asking them to register with the site. The site would automatically connect their accounts with an account of their student organization.
- The leadership of the student body can nominate one 'administrator' for the profile of the student organization on the site. This person woul be responsible for the content of the student organization on the site.
- The 'administrator' would have to fill out a form with detailed questions about the student organization. This information would be used to appropriately categorize student organization, and help users find an organization that best suits their needs.

Exceptional Flow:

- The “unclaimed” profile of a student organization would contain minimum content, which would be obtained from the Student Union. A student organization can “claim” access to their content on the site, by going to the profile of their organization on the site.

Name: Register with the site

Actor: Anyone

Normal Flow:

- User will have to visit the site, and click on ‘register with site.’
- The user will have to either provide their osu email address, or Facebook account to register with the site. An email will be sent to their email address from the site, to confirm their true identity.

Name: Comment on the ‘wall’ of an Event

Actor: Any registered user

Normal Flow:

- The user will have to log onto the site, and navigate to the page with the event details.
- The user can leave a comment on the wall.
- The user can give a ‘thumbs up,’ or ‘thumbs down’ to comments left by other users.

Exceptional Flow:

- If someone has left abusive language or malicious links on the wall, then the site administrators will be able to delete the comment, and reprimand the user.

Name: Add Events on Campus

Actor: Registered users of the site.

Normal Flow:

- The user will have to login to their account, and choose to 'add an event.'
- The user will have to fill information about the event. Some fields would be required, for example name, time, date, location and whether food will be served. The site will ensure that the location information was correctly plotted on the map.
- The user will be able to associate the event to a student organization, as long as the user is an 'administrator' of the student organization or belongs to the 'leadership team' of the student organization.
- All events will be 'public,' i.e. anyone browsing the site will be able to see them.

Exceptional Flow:

- Duplicate events will not be added to the site.

Name: Add buildings/landmarks to the map

Actor: Registered users of the site

Normal Flow:

- Students in Columbus who study at other colleges than OSU, people living in Columbus might be interested in adding landmarks and building names directly to the site. Registered users will be able to add buildings locations on the map. The routing functionality can be then extended to these locations. The locations would also auto-complete in the search boxes.

Name: Help 'clean' schedule and route information

Actor: Registered users of the site.

Normal Flow:

- Some of the route data obtained from screen scrapping COTA's website, was inconsistent. For example, the next figure shows the route of bus #2 is clearly off the road. Users who travel on a route frequently can be used to help 'clean' the data. The registered user will be able to move the location of a bus stop on the map, and make modifications to the structure of the route. If the route it going off the road, the user will be able to pull the line onto the road.

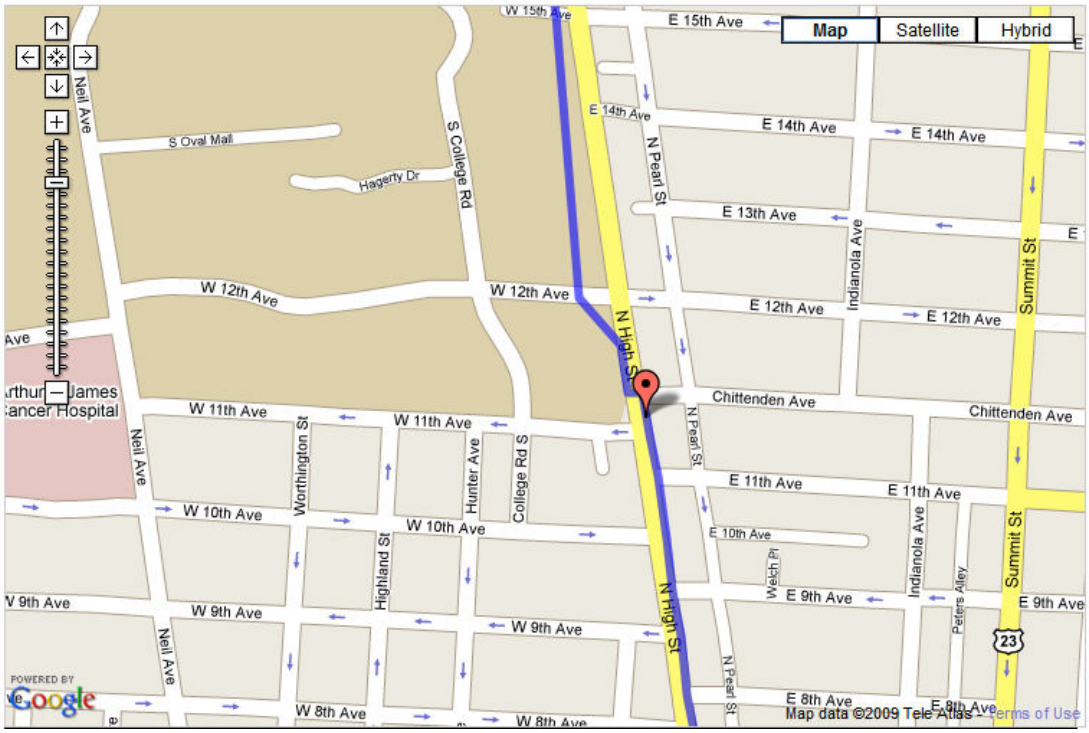


Figure 8: Example of inconsistent data. Route of Bus #2 goes off the road

Name: Maintain a repository of all changes made to the system

Actor: System, Registered users, System Administrator

Normal Flow:

- The system would keep a tab on all modifications made the to the data of the system, which includes changes in bus stop locations, changes in the shape of the route, and even additions of buildings or landmarks to the map.
- System administrator will be able to maintain and reject changes made by users.
- The log file of changes would be visible to registered users.

Name: Provide feedback to changes made by other users

Actor: Registered users of the site.

Normal Flow:

- Registered users will be able to give feedback to specific changes. If a user agrees with a specific change then s/he can give a “thumbs up” to the change; and if the user disagrees with the change then a “thumbs down.”
- The “thumbs up” rating gets associated with the profile of the user. Users with a positive “thumbs up” rating larger than 10 can become the “virtual mayor” of a certain locality. It would be the role of the “virtual mayor” to resolve disputes and maintain data integrity of their location.

Non Functional Requirements

Clean interface

Stimulus Source	End user
Stimulus	User goal: using
Artifact	User Interface
Environment	Runtime
Response	User is able to easily navigate the site. The user takes less than 5 min to complete a use case.
Response Measure	Satisfaction of users, and popularity of the application.
Tactics	• Design time: Separate UI from rest of code. Implement MVC. • Runtime: Collect feedback about usability from end users and implement feedback. • Release the web application in iterations. • Develop a mobile application for the site - as the concept of the application evolved, it was felt that the functionality of the application, such as bus, routing and information of events and organizations should be available on the mobile platform.

End user performance

Stimulus Source	End user
Stimulus	Routing algorithm is fast
Artifact	Normal use

Environment	Runtime
Response	Choose an appropriate routing algorithm.
Response Measure	The user does not have more than 5 seconds to get a response to their request.
Tactics	• Use singleton design pattern; by maintain only one instance of objects that consumed a lot of space in memory, e.g. data of all bus stops location, Dijkstra's graph. Choice of algorithm. • Increase resources – use professional web hosting service of godaddy.com • Reduce resource use: Make optimum number of calls to web-services, and optimize the amount of data transferred. • Have code reviews to promote good coding practices in the team. • Performance impacted the choice of routing algorithm and its implementation.

The application is scalable to include more campuses

Stimulus Source	Developer
Stimulus	Extend the application to more universities, colleges, and public transportation agencies.
Artifact	Database, Performance
Environment	Design time, runtime
Response	• Avoid dependence on non-standard data from third party sources.

	• Structured design of database.
Response Measure	Accommodate for scalability in the design of the database, and the algorithm.
Tactics	• Design database in a manner that more universities, colleges and public transportation agencies can be added with minimal effort. Design the database such that it is general, and not customized to Columbus and The Ohio State University campus. • It influenced the choice of routing algorithm. Initial approach of using adjacency matrices to find optimal routes was dropped, as it used about 6 GB of memory for 1 city, and took a very long time to pre-compute all possible routes. Instead Dijkstra's algorithm was adopted, which uses significantly less memory. It uses about 100 MB in memory, and the graph is not written to disk. As all possible routes are not pre-computed by Dijkstra's algorithm the amount of time used to create the Dijkstra's graph is the order of a few minutes for one city. • Application depends on Google Transit Feed Specification for routing over public transit systems. This specification being used by a large number of agencies in the US and in other countries of the world. • Second source of data for the application is RSS feeds for 'events.' Design the code such that it decouples the data source and the database of the system.

Code is well tested

Stimulus Source	Developer
Stimulus	Routing algorithm deals with a lot of data and is complex. There needs to be a repeatable way to test different components of the code at different stages of development.
Artifact	Code, algorithm
Environment	Implementation
Response	Write automated test cases for different components and the entire system.
Tactics	• Write JUint tests for testing Classes and complex methods. • Write JUint System tests. • Code reviews with the Professor.

CHAPTER 3

Database

The process of going from raw data to a route involves three steps. First, the data from public transportation agencies, which is in Google Transit Feed Specification (GTFS) format, is converted into a MySQL database. This data is then used to create Dijkstra's graph in memory. Bus stop locations at a specific point in time act as nodes of the Dijkstra's graph. The nodes are connected together with bus edges, waiting edges and walking edges. This Dijkstra's graph is kept in memory the entire time. Finally, when a web-service is called to find a route between two locations (and a specific time in the day), the Dijkstra's graph is solved to find the optimal route.

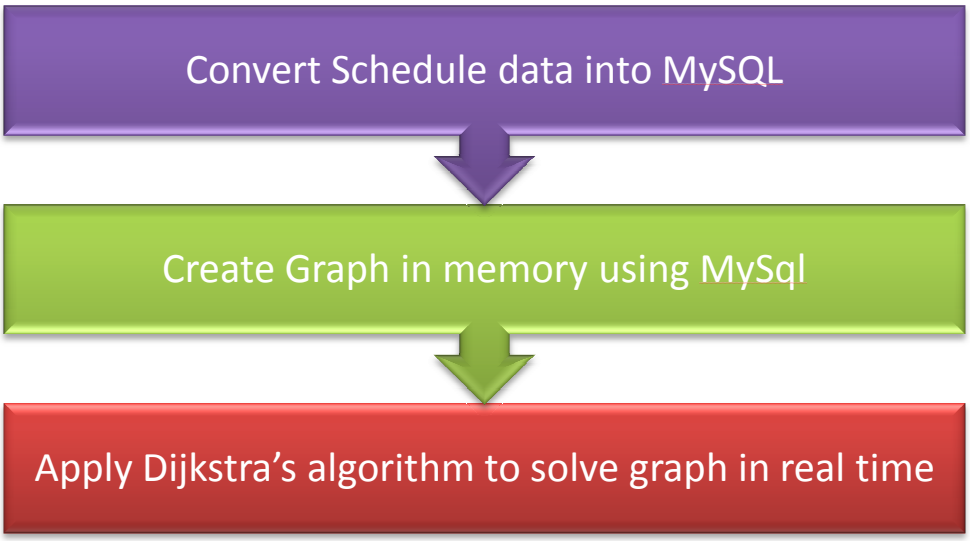


Figure 9: Steps involved

Database Schema

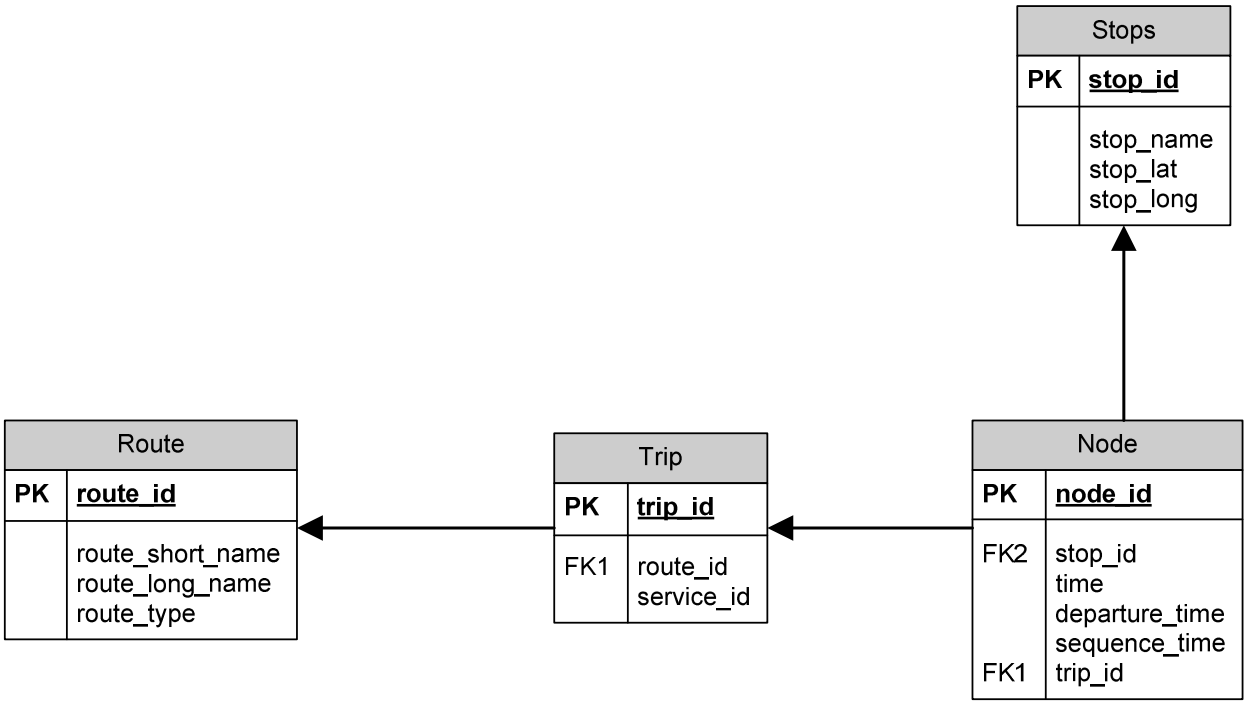


Figure 10: Entity Relationship Model

The database schema used for a MySQL database is given below. Primary keys have been underlined, whereas attributes with foreign key constraints have been highlighted. Each of these terms is explained in the next part of the Chapter which focuses on explaining the basic structure of GTFS.

- Stops (stop_id, stop_name, stop_lat, stop_lon)
- Routes (route_id, route_short_name, route_long_name, route_type);
- Trips (trip_id , route_id, service_id)
- Nodes (id, trip_id, stop_id, time, departure_time, stop_sequence)

Google Transit Feed Specification (GTFS)

GTFS defines a common format for public transportation schedules and associated geographic information (Google, 2009). It consists of 12 comma separated values (csv) files, created by the public transportation agency to provide schedules and geographic information to applications like Google Maps. The following four files are relevant to the get an understanding of the implementation of the routing algorithm. The required data fields in the files have been explained below.

Field Name	Details
stop_id	An ID that uniquely identifies a stop or station. Multiple routes may use the same stop. The stop_id is dataset unique.
stop_name	The name of a stop or station. A name that people understand in the local and tourist vernacular.
stop_lat	The latitude of a stop or station.
stop_lon	The longitude of a stop or station.

Table 1: Explanation of stops.txt in GTFS

Field Name	Details
route_id	An ID that uniquely identifies a route.
route_short_name	The short name of a route.
route_long_name	The full name of a route.

route_type	Describes the type of transportation used on a route. Valid values for this field are: 0 - Tram, Streetcar, Light rail. Any light rail or street level system within a metropolitan area. 1 - Subway, Metro. Any underground rail system within a metropolitan area. 2 - Rail. Used for intercity or long-distance travel. 3 - Bus. Used for short- and long-distance bus routes. And so on…

Table 2: Explanation of routes.txt in GTFS

Field Name	Details
route_id	An ID that uniquely identifies a route. This value is referenced from the routes.txt file.
service_id	An ID that uniquely identifies a set of dates when service is available for one or more routes. This value is referenced from the calendar.txt or calendar_dates.txt file. (For the scope of the project, this field was ignored.)
trip_id	An ID that identifies a trip. The trip_id is dataset unique.

Table 3: Explanation of trips.txt in GTFS

<table>
<tr><th>Field Name</th><th>Details</th></tr>
<tr><td>trip_id</td><td>An ID that identifies a trip. This value is referenced from the trips.txt file.</td></tr>
<tr><td>arrival_time</td><td>Arrival time at a specific stop for a specific trip on a route. The time is measured from midnight at the beginning of the service date. For times occurring after midnight on the service date, enter the time as a value greater than 24:00:00 in HH:MM:SS local time for the day on which the trip schedule begins.
For example,
<table>
<tr><td>Time</td><td>arrival_time value</td></tr>
<tr><td>08:10:00 A.M.</td><td>08:10:00 or 8:10:00</td></tr>
<tr><td>01:05:00 P.M.</td><td>13:05:00</td></tr>
<tr><td>07:40:00 P.M.</td><td>19:40:00</td></tr>
<tr><td>01:55:00 A.M.</td><td>25:55:00</td></tr>
</table>
</td></tr>
<tr><td>departure_time</td><td>The departure time from a specific stop for a specific trip on a route.</td></tr>
<tr><td>stop_id</td><td>An ID that uniquely identifies a stop. Multiple routes may use the same stop. The stop_id is referenced from the stops.txt file.</td></tr>
<tr><td>stop_sequence</td><td>Identifies the order of the stops for a particular trip.</td></tr>
</table>

Table 4: Explanation of stop_times.txt in GTFS

CHAPTER 4

Dijkstra's Algorithm

Explanation

Dijkstra's algorithm is a single source shortest path algorithm for a weighted directed graph G = (V, E) (Cormen, 2003). Vertices in the graph are connected by edges. Each edge has a weight associated with it. G has all edge weights that are nonnegative, hence,

$$w\ (u, v) \geq 0\ for\ each\ edge(u, v) \epsilon\ E$$

In the case of a road network, the weight may be the distance between two vertices. None of the edge weights in the graph are negative. Dijkstra's algorithm finds the shortest path between a given vertex and all the other vertices in the graph. The algorithm starts from one vertex, and extends outwards in the graph, at each stage adding the vertex to the graph which has the least distance (weight) from the source. The process repeats itself till all the vertices in the graph have been reached, or when no other vertex in the graph can be reached by the expanding Dijkstra's tree (Dijkstra's Algorithm, 2009). The algorithm is an example of a greedy algorithm, which means, that when the algorithm is extending outwards in the graph, it adds the vertex with the smallest weight first to the graph.

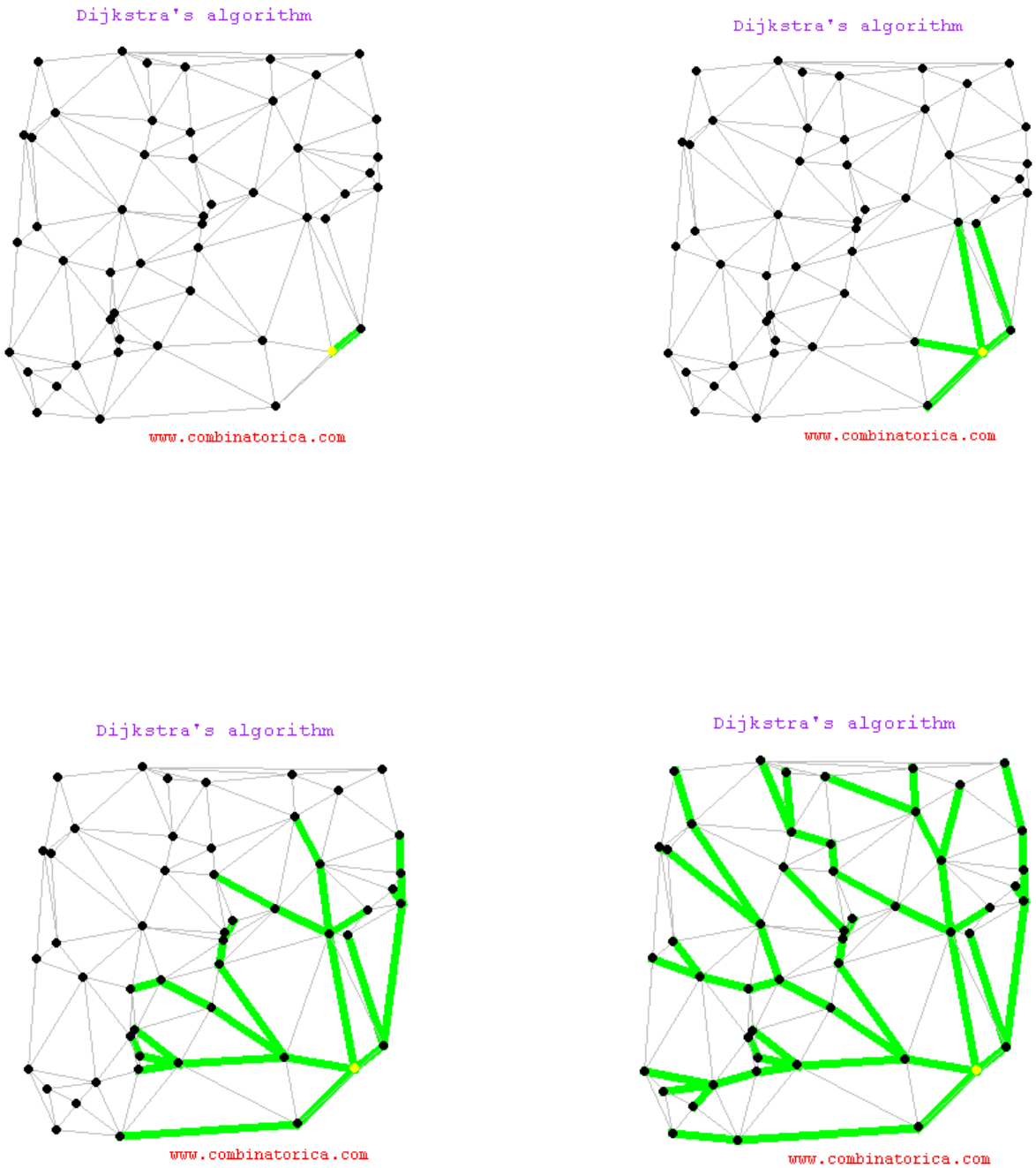


Figure 11: Dijkstra's Algorithm (Skiena)

Correctness of Dijkstra's algorithm

If Dijkstra's algorithm is run on a weighted, directed graph G = (V, E) with non negative weight function w and source s, then at termination, $distance[s,u] = \delta(s,u)$, for all vertices $u \in V$, (Cormen, 2003)

where, $distance[s,u] = summation\ of\ edge\ weights\ from\ source\ s\ to\ node\ u, and$

$\delta(s,u) = minimum\ edge\ weight\ to\ get\ from\ s\ to\ u, if\ s\ and\ u\ are\ connected$

$= \infty, otherwise.$

Priority Queue implementation

procedure Dijkstra_Shortest_Path(G, s)

/* construct shortest path from v_s to all vertices of G */

/* Q is a priority queue of vertices */

1. **for** each vertex $v_i \in$ V(G) – $\{v_s\}$ **do**
2. Q.insert (v_i , ∞) ;
3. Q.insert (v_s , 0) ;
4. v_s. parent ← NIL ;
5. v_s. distance ← 0 ;
6. **while** Q.IsNotEmpty() and Q.GetMinKey() ≠ ∞ **do**
7. v_j ← Q.DeleteMin();

   /* shortest path edge = (v_j.parent, v_j) */
8. **for** each edge (v_j , v_k) incident on v_j **do**
9. new_dist ← v_j.distance + weight(v_j , v_k) ;
10. **if** Q.Contains(v_k) and new_dist < Q.Key(v_k) **then**
11. Q.DecreaseKey(v_k , new_dist) ;
12. v_k.parent ← v_j ;

13. v_k.distance ← new_dist;

Above is pseudo code implementation of Dijkstra's shortest path algorithm using a priority queue. G is the graph, and V(G) represents the set of all vertices in the graph G. v_s is the source vertex. In the first three lines the priority queue(Q) is initialized. Q represents a pairing of a vertex in the graph to the corresponding distance of the vertex from the source vertex (v_s). Initially all vertices are at ∞ from v_s, expect v_s itself, which is at a distance 0 from itself. v_s has no parent in the Dijkstra's tree that is being constructed. From line 6 onwards, the algorithm enters a loop. The vertex with the shortest distance from the v_s is deleted from Q. Each edge(v_j , v_k) going out of Q is analyzed. If v_k, is contained in Q (i.e. if , v_k has not been reached by the tree), and if the distance from v_s to v_k is shorter than the distance that exists in Q for , v_k, then the distance in Q is updated with the smaller value and the parent of , v_k is updated. The loop repeats itself, each time pulling out a vertex, v_j from Q which is the next closet vertex to v_s, and looking at each of the edges of v_j. The loop exists when Q is empty, or when all vertices in Q cannot be reached by the Dijkstra's graph (i.e. the vertices are at ∞ distance from all vertices in the Dijkstra's graph).

Complexity

For a priority queue Q, of size s, the following table gives the complexity of various operations that belong to the priority queue class.

Table 5: Complexity of operations belonging to priority queue

Operation	Complexity
Q.Insert(Object x, Key k)	Θ(log s)
X ←Q.DeleteMin()	Θ(log s)
Q.IsEmpty()	Θ(1)
Q.DecreaseKey(Object x, Key k)	Θ(log s)
Q.IsNotEmpty()	Θ(1)
Q.Contains(Object x)	Θ(1)
K ← Q.Key(Object x)	Θ(1)

If n is the number of vertices, and m is the number of edges in the graph, then the following table gives the complexity of the Dijkstra's algorithm.

procedure Dijkstra_Shortest_Path(G, s)

/* construct shortest path from v_s to all vertices of G */

/* Q is a priority queue of vertices */

1. **for** each vertex $v_i \epsilon$ V(G) – $\{v_s\}$ **do**
2. Q.insert (v_i , ∞) ;
3. Q.insert (v_s , 0) ;

} $\sum_{s=1}^{n} (\log s) + c = n \log n + c$

4. v_s. parent ← NIL ;
5. v_s. distance ← 0 ;

} constant

6. **while** Q.IsNotEmpty() and Q.GetMinKey() ≠ ∞ **do**
7. v_j ← Q.DeleteMin();

} $\sum_{s=1}^{n} (\log s) + c = n \log n + c$

/* shortest path edge = (v_j.parent, v_j) */

8. **for** each edge (v_j , v_k) incident on v_j **do**
9. new_dist ← v_j.distance + weight(v_j , v_k) ;
10. **if** Q.Contains(v_k) and new_dist < Q.Key(v_k) **then**
11. Q.DecreaseKey(v_k , new_dist) ;
12. v_k.parent ← v_j ;
13. v_k.distance ← new_dist;

} $\sum deg(v_j)$

} $\sum deg(v_j) \log n + c$

← $\log n$

Therefore total complexity of the algorithm is $\Theta(\, n \log n + m \log n\,)$.

CHAPTER 5

Application of Dijkstra's Algorithm

Time Extended Dijkstra's Algorithm

Graph for a road looks very different from a graph that has been made from bus schedules. A graph for a road network is 2-dimensional. The dimensions are latitude and longitude. Below is a graph of a road network which connects Columbus Downtown to the OSU campus, and then to the airport. No matter at what time of the day the journey is made, the same road network can be used to go from Downtown to the University and then to the airport.

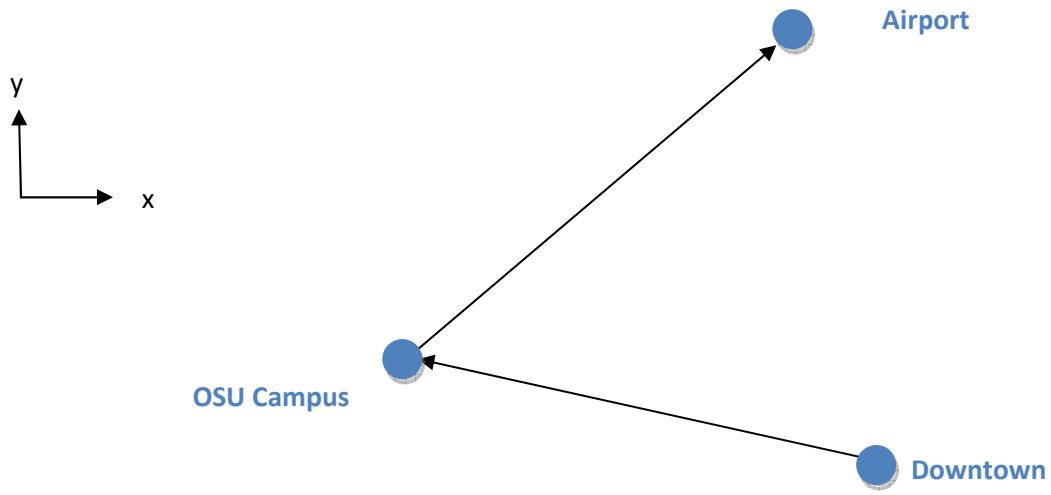


Figure 12: Graph for a road network

However, the same is not true for a graph made from bus schedules. This graph is a 3-dimensional graph. The dimensions are latitude, longitude and time. If a bus leaves downtown at 8.00 am, it reaches the University at 8.15 am. If the only next bus leaves downtown at 9.00 am, it reaches the University at 9.15 am. Now, if someone wants to leave Downtown at 8.30 am, the person cannot get onto a bus immediately. The person will have to wait till 9.00 am to take the bus to the University. The next figure shows such a graph made from bus schedules. It is almost as if a number of road network graphs are stacked on one top of the other. It should be noted, that these graphs stacked on top of the other need not be the same. In most cases they are different and that is because of the following reasons.

1. There are a number of bus routes that serve a particular bus stop.
2. The bus stop is served by routes that operate at different frequencies throughout the day.
3. Route of a particular bus can be different during different times of the day.

A node in the graph on the next page can be written as a function of the 3-dimensions - Node(latitude, longitude, time). On the other hand a Stop, such as OSU campus, has 2-dimensions - Stop(latitude, longitude).

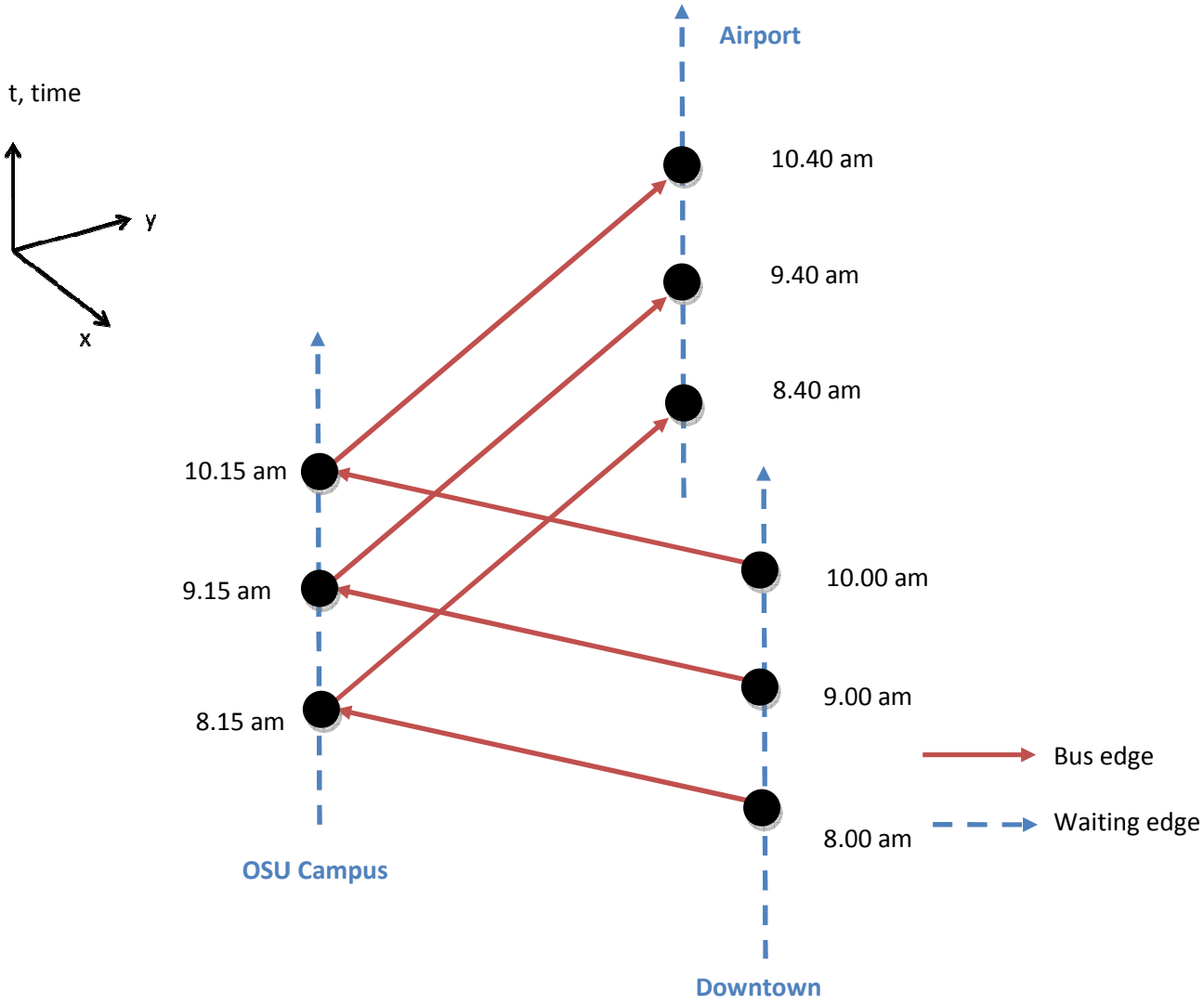


Figure 13: Graph for bus schedules

Edges

There are three types of edges, and they are all handled differently by the algorithm. The three edge types are bus, waiting and walking edges.

1. **Bus edges** – edges that are formed when a bus goes from one node to the other. In the figure below, the red arrow shows a bus edge between node(lat1, lon1, t1) and node(lat2, lon2, t2).

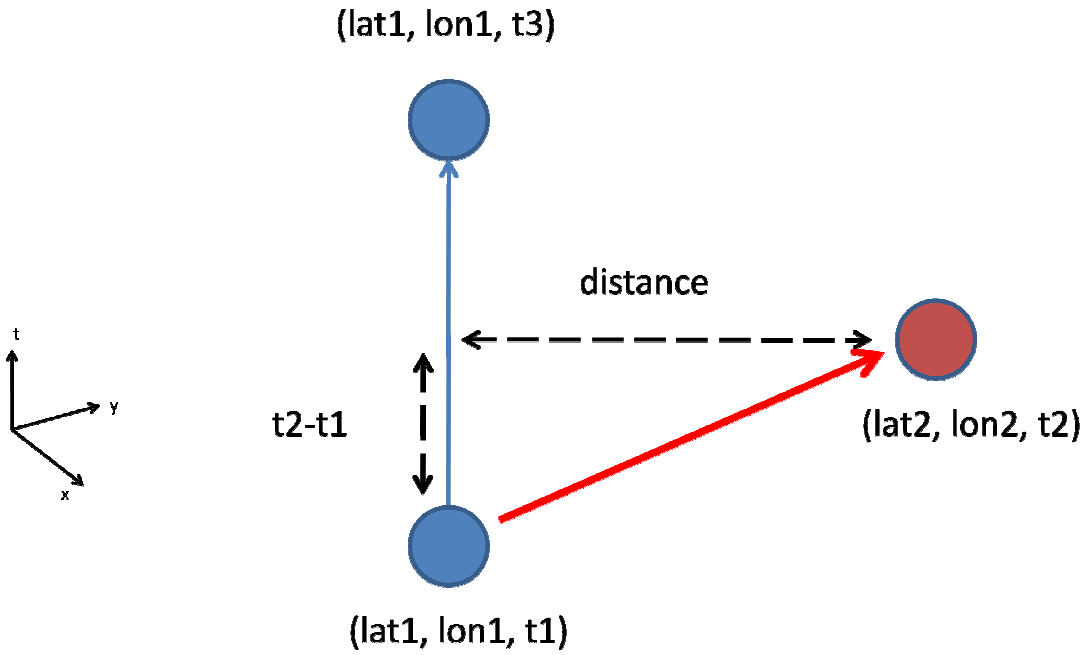


Figure 14: Bus edge and distance between bus stops

'Great circle distance' formula is used to calculate the distance between the two bus stops (Meridian World Data, 2009). The great circle distance calculates the shortest distance between two points on a spherical surface. Bus stops are located on a spherical earth and hence the great circle distance formula is used to calculate the distance between two bus stops. The formula is as follows.

$$\boldsymbol{distance} =$$

$$\boldsymbol{r} * \mathbf{arccos}\ (\mathbf{sin}(\boldsymbol{lat1}) * \mathbf{sin}(\boldsymbol{lat2}) + \mathbf{cos}(\boldsymbol{lat1}) * \mathbf{cos}(\boldsymbol{lat2}) * \mathbf{cos}(\boldsymbol{lon2} - \boldsymbol{lon1}))$$

Where r $=$ radius of the earth

$= 3963.5\ (miles)\ or\ 6378.7\ (km)$

To use the formula, latitude and longitude need to be in radians.

2. **Waiting edges** – This edge connects two nodes, which reference the same bus stop. This edge is formed when one simply waits at a bus stop. The 'waiting' edge is formed between two consecutive nodes of the same bus stop, when they are sorted according to time.

For example, one bus stop, say Neil & 5th Avenue, is served by two bus routes, one heading towards High Street and 5th Avenue and the other towards Neil and 10th Ave. In the figure below, nodes of the graph are shown to be of different sizes to represent a three dimensional space. A, B and C are 3 nodes of the same bus stop – Neil & 5th Avenue. It should be noted that a 'waiting' edge exists only between A and B, and another between B and C, even though the pair of nodes cater to different bus routes. A and C are not connected together by a 'waiting' edge.

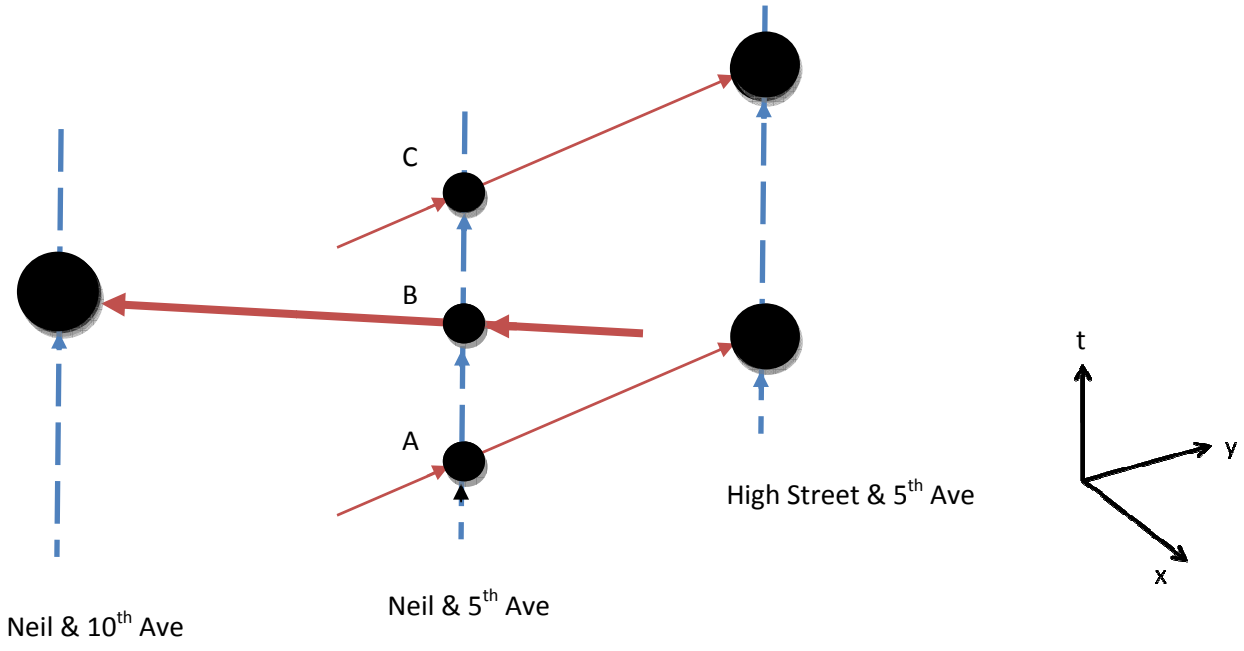


Figure 15: Example of 'waiting' edges

3. **Walking edges** – These edges connect nodes of two different bus stops which are close enough such that a user could walk from one bus stop to the other. No bus is taken to make this journey.

In this example, 'walking' edges for the node A are explored. A is connected to node B by a 'bus' edge, and to node C by a 'waiting' edge. Suppose MAX_WALKING_DISTANCE is the maximum amount of distance the user is ready to walk from one bus stop to another and WALKING_SPEED is the walking speed of the user. (Value of WALKING_SPEED can be set to a value less than the average walking speed of a person.) Plotting this information on the graph generates a figure consisting of a green cone and a cylinder (as shown in the following figure with the green dashed lines). The vertex of the cone is node A. All nodes which fall inside the volume of this figure are reachable from node A by walking. However, this figure also will contain nodes which are separated from A by a big time difference (say for more than an hour). It is impractical to expect a user to wait at a bus stop for a long time, e.g. over an hour. Hence, a constant MAX_WALKING_TIME is used to set an upper limit to height of the figure (the combination of the cone and the cylinder). Any nodes that fall within this figure, have walking edges with A. In other words, a user will be able to walk from node A to node D, because the speed required to cross the distance is less than the WALKING_SPEED, and the time difference that exists between nodes D and A is less than MAX_WALKING_TIME.

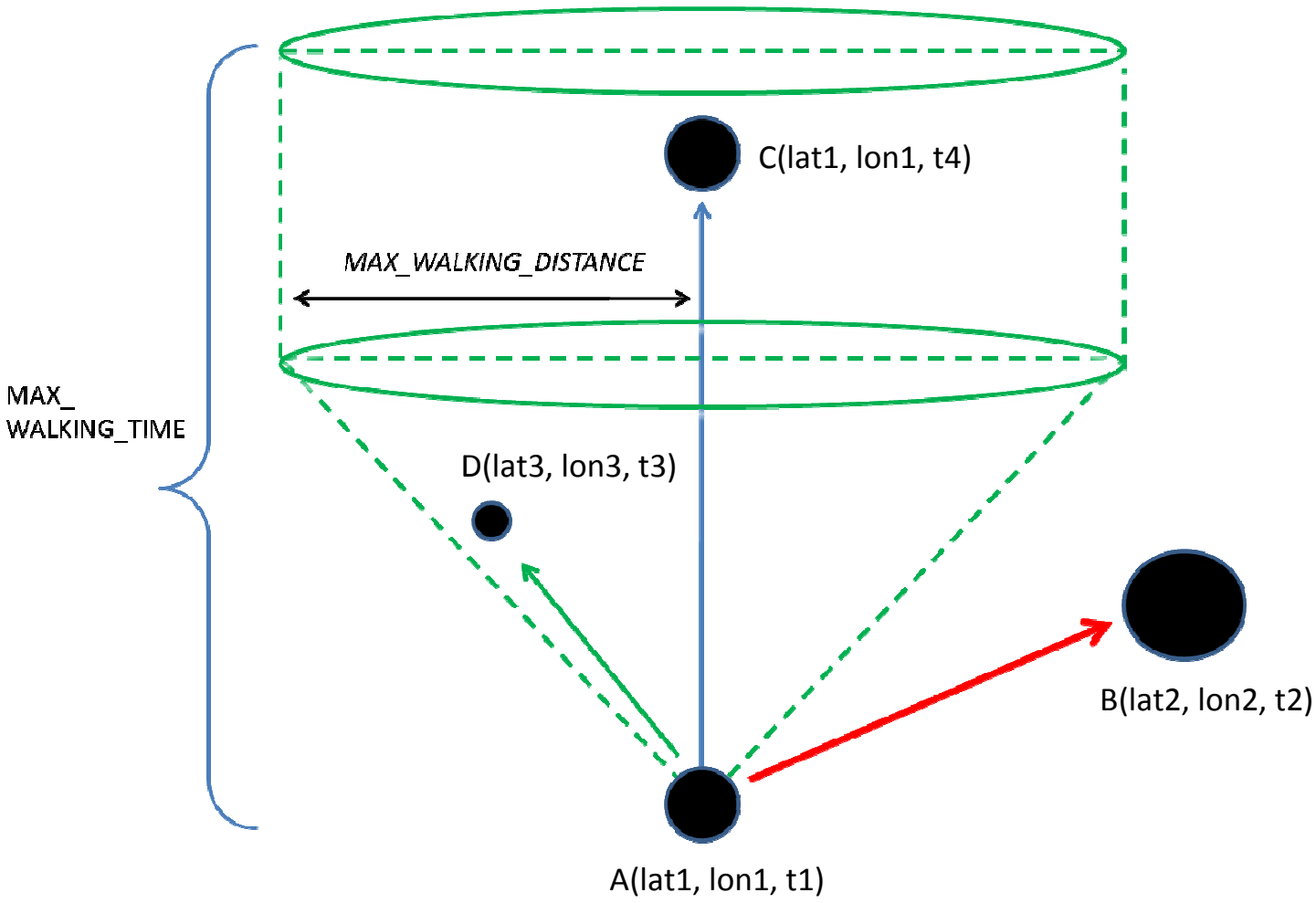


Figure 16: Walking Edges in Graph

Naïve implementation for adding walking edges

Pseudo-code for adding walking edges is as follows.

1. $\forall\, v_i \,\epsilon\, V(G)$
2. $\quad \forall\, v_j \,\epsilon\, V(G) - \{v_i\}$
3. $\quad\quad if(distance\,(v_i, v_j) < MAX\_WALKING\_DISTANCE\,)$
4. $\quad\quad\quad time\_required \;= \frac{distance\,(v_i, v_j)}{WALKING\_SPEED};$
5. $\quad\quad\quad time_{diff} = \; v_j.time - v_i.time;$
6. $\quad\quad\quad if(\,0 < time_{diff} < MAX\_WALKING\_TIME)$
7. $\quad\quad\quad\quad if\;(time\_required \quad < time_{diff})$
8. $\quad\quad\quad\quad\quad add\ walking\ edge\ between\ v_i\ and\ v_j;$
9. $\quad\quad\quad\quad end\ if$
10. $\quad\quad\quad end\ if$
11. $\quad\quad endif$
12. $\quad end\ for$
13. $end\ for$

The complexity of the algorithm is θ (n^2), where n is the number of vertices (or nodes) in the graph. A city like Columbus has over 0.25 million nodes, and Austin has over 0.56 million nodes. Add walking edges is an expensive process. Walking edges (like other edges) are pre-computed. Once created, the graph is kept in memory the entire time, and solved in real time by Dijktra's algorithm. Hence the running time of the algorithm mentioned above does not negatively affect the running time of Dijkstra's algorithm. However, it takes too long to add

walking edges to the Dijkstra's graph using the naïve implementation as the following analysis proves.

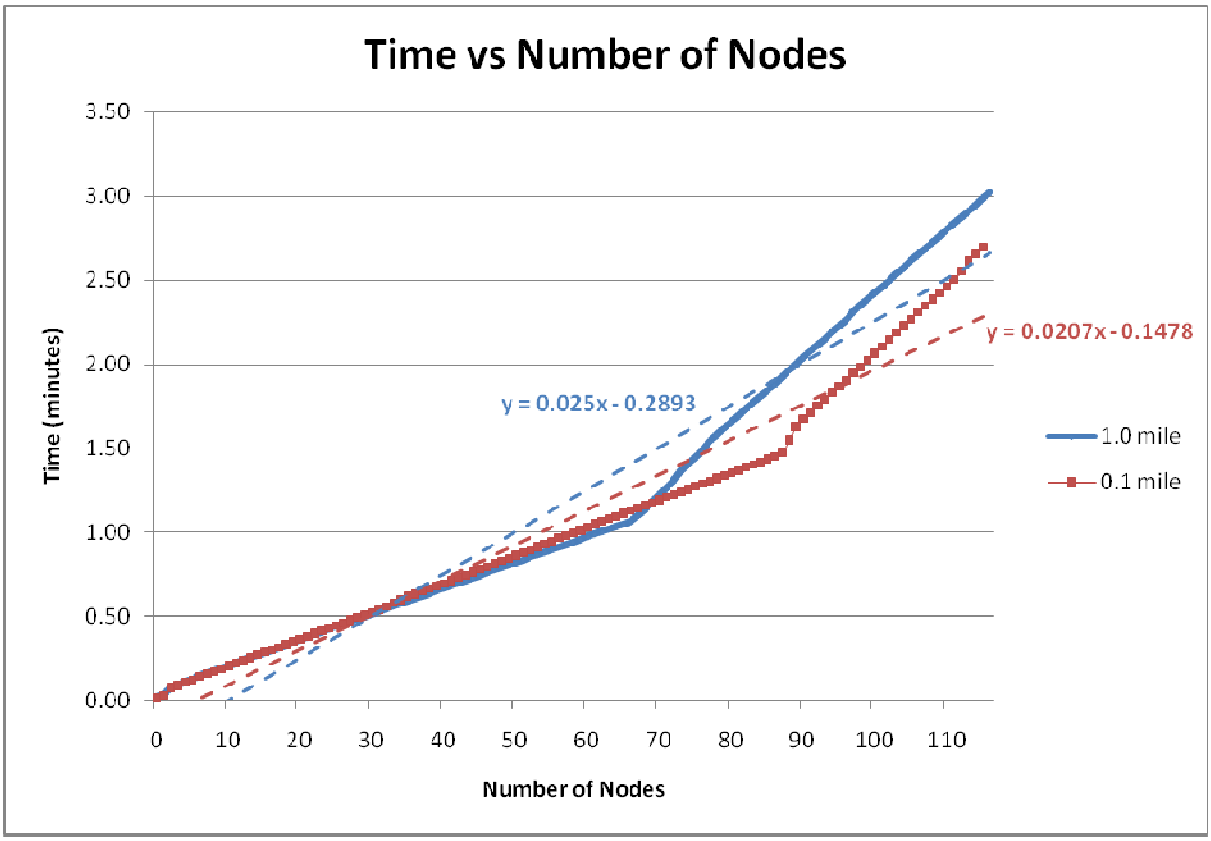


Figure 17: Graph of Time vs Number of Nodes for adding walking edges by naive implementation

The graph above shows the running time of the naïve implementation of adding walking edges to the Dijkstra's graph for the city of Austin. The Dijkstra's graph for city of Austin has 560,225 nodes. The graph above shows two solid lines and two dashed lines. The solid red line represents the running time of the algorithm where MAX_WALKING_DISTANCE = 0.1 mile, and the solid blue line represents the algorithm with the value of 1.0 mile. More combinations of nodes need to be compared if the MAX_WALKING_DISTANCE is larger, and hence as expected the blue line takes more time than the red line to process 116 nodes. The two dashed lines are

trend lines, one for each of the solid lines. A linear equation representing the two trend lines is also shown in the graph. Approximating that the time it takes to process the nodes is linear, it will take 8.05 days to add walking edges between bus stops that are at a distance of 0.1 mile from each other. Moreover, it will take 9.73 days to add walking edges between bus stops that are at a distance of 1.0 mile from each other. This method might not be the most accurate way of predicting how long the algorithm will take to add walking edges to the graph, but it does tell us that it takes about a week (or more) to add walking edges for a city that is the size of Austin. Clearly, it takes too long to add walking edges to the Dijkstra's graph for only one city, and hence this algorithm is not scalable if more cities need to be added to the project.

Efficient Algorithm for Adding Walking Edges

In the algorithm below, S is the set of bus stops in the city, and 'map' is an instance of a Map object with (key, value) pairs which map Stops to List of Nodes that they belong to the stops.

1. $\forall\, s_i \in S$
2. $\quad Sort\_By\_Time(map.getValue(s_i);$
3. $end\ for$
4. $\forall\, s_i \in S$
5. $\quad \forall\, s_j \in S - \{s_i\}$
6. $\quad\quad if(distance\ (s_i, s_j) < MAX\_WALKING\_DISTANCE\ )$
7. $\quad\quad\quad List < Node > \ L1 = map.getValue(s_i);$
8. $\quad\quad\quad List < Node > \ L2 = map.getValue(s_j);$
9. $\quad\quad\quad List < Node > \ L = Merge\_By\_Time(L1, L2);$
10. $\quad\quad\quad time\_required \ = \frac{distance(s_i, s_j)}{WALKING\_SPEED};$
11. $\quad\quad\quad for\ (m = 0;\ m < L.size() - 1;\ m++)$
12. $\quad\quad\quad\quad node1 \ = \ L.get(m);$
13. $\quad\quad\quad\quad for\ (h = m + 1;\ h < \ L.size();\ h++)$
14. $\quad\quad\quad\quad\quad node2 \ = \ L.get(h);$
15. $\quad\quad\quad\quad\quad if\ node1.getStopId() \neq node2.getStopId()$
16. $\quad\quad\quad\quad\quad\quad diff \ = \ node2.getTime() \ - \ node1.getTime();$
17. $\quad\quad\quad\quad\quad\quad if\ time\_required < \ diff < \ MAX\_WALKING\_TIME$
18. $\quad\quad\quad\quad\quad\quad\quad add\_walking\_edge(node1, node2);$
19. $\quad\quad\quad\quad\quad\quad\quad break;$

20. $end\ if$

21. $end\ if$

22. $end\ for$

23. $end\ for$

24. $end\ for$

25. $end\ for$

Explanation

The algorithm starts off by sorting the map, according to the time of the node; i.e. for each bus stop, the corresponding list of nodes is sorted in increasing time of the nodes. The algorithm then compares each bus stop with another. If the distance between the two bus-stops is less than the MAX_WALKING_DISTANCE then the execution reaches line 7. The list of nodes for each bus stop is obtained and merged into a new list of nodes, according to the time of the nodes. This is a linear operation that is proportional to the length of the new list L. Time required to walk between the two bus stops is calculated. List L is traversed to find pairs of nodes belonging to different bus stops that the

- time difference such it is more than the required time to walk between the two bus stops, and
- time difference is less than the MAX_WALKING_TIME.

For each pair of nodes that meets these criteria a walking edge is added between the nodes. An example is illustrated on the next page.

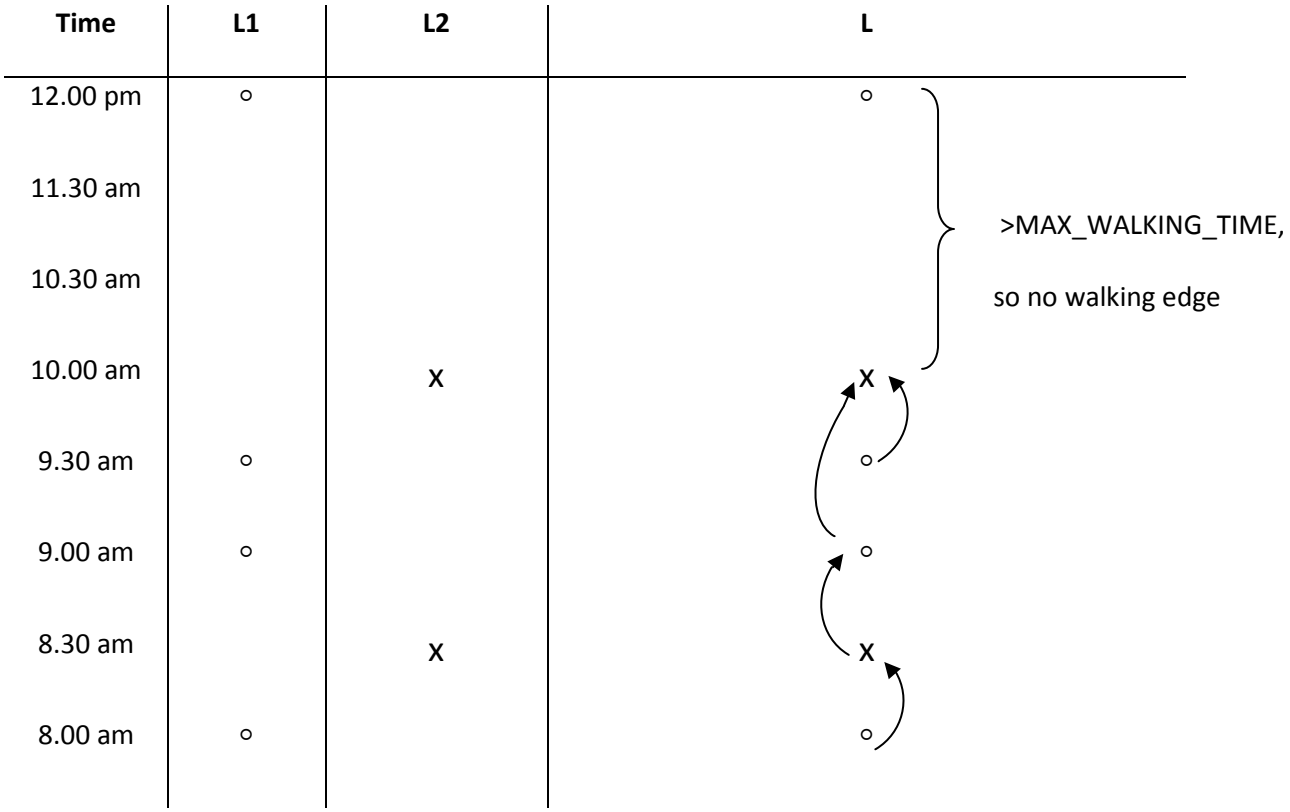


Figure 18: Example of walking edges

Complexity

Say there are n number of nodes in the graph, and s number of total bus stops in the graph. Also, assuming that on average there are $^{n}/_{s}$ number of nodes per bus stop. Hence, the time it takes to sort the 'map' object (Mapping between bus stops and nodes that serve these bus stops) $= s * \frac{n}{s} * \log\left(\frac{n}{s}\right) = n\ (\log n - \log s\ ) =\ \theta\ (n \log n)$

(s << n; s is of the order of 3010 bus stops, where n is of the order of 560,000 points).

The insides of the two outer for loops (starting at lines 4-5) are executed only when the two bus stops being considered are close to one another such that the distance between them is less than the MAX_WALKING_DISTANCE. Thus the loops are only executed $^{s^2}/_{k}$ times, where k depends on MAX_WALKING_DISTANCE.

Line 9, merges two Lists of nodes according to time of the nodes. This line takes time proportional to the addition of the lengths of the two lists, i.e. proportional to $^{2 * n}/_{s}$. The inner loops (line 11-20) also are proportional to the addition of the lengths of the two lists, i.e. proportional to $^{2 * c * n}/_{s}$.

Therefore, the efficient algorithm to add walking edges to the graph takes time

$$= n * \log n + \frac{s^2}{k} * \left(\frac{2 * n}{s} +\ 2 * c * \frac{n}{s}\right)$$

$$= n * \log n + \frac{c` * n * s}{k}$$

$$=\ \theta\left(n * \frac{s}{k}\right), \qquad (as \log n \ll s)$$

If value of MAX_WALKING_DISTANCE is large then the value of 1/k is very large, and hence the running time of the algorithm becomes very large. The relationship between 1/k and MAX_WALKING_DISTANCE has been shown on the graph on the next page. However, if the MAX_WALKING_DISTANCE is 0.15 miles or lower the running time of the algorithm is less than one minute, considerably quicker than the naïve implementation, which would have taken approximately 10 days.

$$k = \frac{Total\ number\ of\ bus\ stops}{number\ of\ walking\ edges}$$

City	Number of Nodes, n	Number of bus stops, s
Austin	560,225	3,010
Columbus	274,426	4,260

Table 6: Number of nodes and stops for Austin and Columbus

For the city of Austin the number of walking edges jumps from 0 to 524 when MAX_WALKING_DISTANCE increase from 6.76 m to 6.92 m. This is most likely due to the fact that bus stops on the opposite site of the road get considered by the algorithm when the MAX_WALKING_DISTANCE is 6.92 m. For the city of Columbus, even when the MAX_WALKING_DISTANCE is 0 m there are 198 walking edges. This is because there are different bus stops with the same latitude and longitude coordinates in the database. This could be due various reasons, including inaccuracies in the database, or even inaccuracies in measuring the exact location of a bus stop.

MAX_WALKING_DISTANCE		Austin		Columbus	
(miles)	(meters)	Number of walking edges	1/k	Number of walking edges	1/k
0.0000	0.00	0	0.00	198	0.05
0.0025	4.02	0	0.00	700	0.16
0.0040	6.44	0	0.00	776	0.18
0.0042	6.76	0	0.00	776	0.18
0.0043	6.92	524	0.17	776	0.18
0.0044	7.08	524	0.17	776	0.18
0.0045	7.24	524	0.17	776	0.18
0.0050	8.05	524	0.17	986	0.23
0.0100	16.09	26,572	8.83	32,956	7.74
0.0300	48.28	604,836	200.94	297,498	69.84
0.0500	80.47	948,158	315.00	460,646	108.13
0.0800	128.75	1,811,006	601.66	786,290	184.58
0.1000	160.93	2,497,144	829.62	1,173,820	275.54
0.1300	209.21	3,801,414	1262.93	1,807,454	424.28
0.1500	241.40	5,001,038	1661.47	2,356,842	553.25
0.1800	289.68	Memory crash!		3,156,152	740.88
0.2000	321.87			3,754,532	881.35
0.2200	354.06			4,298,350	1009.00
0.2400	386.24			4,920,654	1155.08
0.2600	418.43			5,569,790	1307.46
0.2800	450.62			Memory crash!	

Table 7: Number of walking edges and value of 1/k for different values of MAX_WALKING_DISTANCE

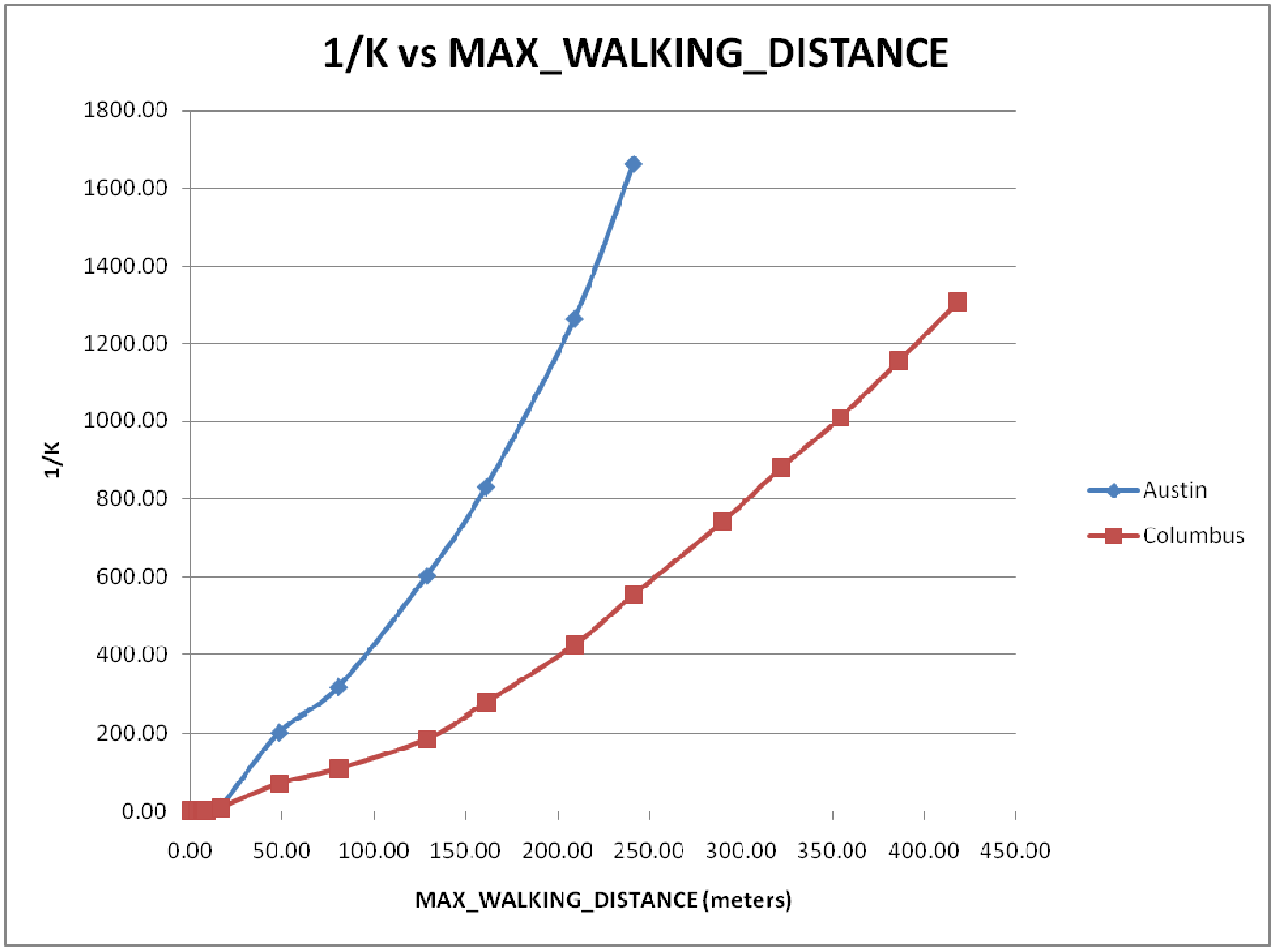
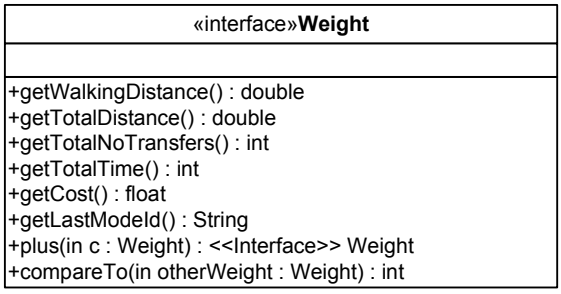


Figure 19: Graph depicting the relationship between 1/K and the MAX_WALKING_DISTANCE

Weights

When a user is taking a bus their main concern is time and convenience of the journey. The user prefers to travel on the route which takes the shortest time. However, if the shortest route involves taking multiple bus transfers then the user will not prefer to take the route, especially if the time saved is not significant. Moreover, a user might prefer to take a route which involves less walking even if means additional waiting at the bus stop to catch another bus. A passenger has to weight different criteria before picking the most convenient route. The criteria include (but are not limited to) the total time of the journey, number of bus transfers required to complete the journey, and amount of distance required to walk in order to catch a transfer.

In order to accurately represent the decision making of a user, the algorithm associates different weights (or cost) with the different edge types. The Weight interface is shown below. The implementation of the interface keeps a track of different fields, such as the total walking distance, total number of transfers and total time of journey, among others. Dijkstra's implementation of the shortest path algorithm does not know of these criteria. It uses weight (or cost) as any other conventional implementation of Dijkstra's algorithm would. It simply makes a call to add two weights together, and the priority queue uses the *compareTo()* method to compare two different weights.



Addition of Weights

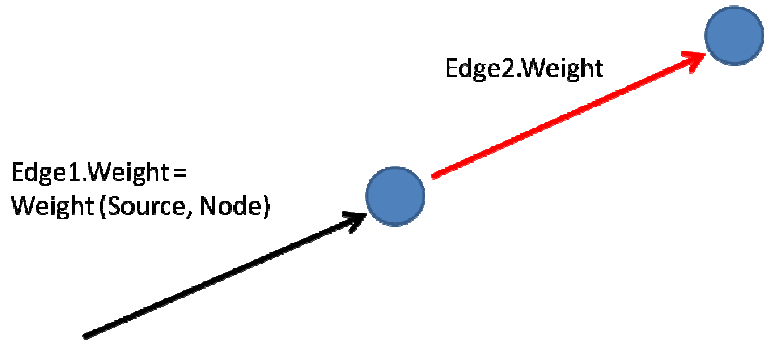


Figure 20: Addition of two weights

Dijkstra's algorithm has to decide which node to add to the expanding tree only by looking at two pieces of data for all the nodes in consideration. These are

1. the weight of Edge1, which is the sum of the edge weights from the source to the node under consideration, and
2. weight of Edge2

Below, are some rules that are used in the implementation of the 'Weight' interface.

1. A situation where the user is travelling on a bus, which approaches the bus stop and then continues on with its journey. The user continues to stay on the bus.

$$i.e. if\ Edge1.Type = Edge2.Type = Bus, \quad and \quad Edge1.route_{id} = Edge2.route_{id},$$
$$then$$
$$Edge1.weight + Edge\ 2.weight = weight\ object$$

2. The user cannot be expected to get off the bus and get onto another bus at the very same instant of time. The algorithm in such a circumstance will consider this situation to be impossible and will set the weight to infinity.

$$i.e. if\ Edge1.Type = Edge2.Type = Bus, \quad and \quad Edge1.route_{id} \neq Edge2.route_{id},$$
$$then$$
$$Edge1.weight + Edge\ 2.weight = \infty$$

3. The user gets off a bus and walks to another bus stop or continues to wait at the bus stop. In such a scenario, the algorithm will simply add different criteria and return a weight object.

$$i.e. if\ Edge1.Type = Bus, \quad and\ Edge2.Type = Walking\ or\ Waiting, \quad then$$
$$Edge1.weight + Edge\ 2.weight = Weight\ object$$

4. The user walks to a bus stop or has been waiting at a bus stop. A bus approaches and the user gets onto the bus. If the user has not taken any transfers before this time period, then it means that this is the first bus that the user is getting on. Hence a weight object is created

and returned. However, if the user has already taken transfers in the journey, then the number of transfers is also incremented by 1.

$$i.e. if\ Edge1.Type = Walking\ or\ Waiting, \quad and\ Edge2.Type = Bus, \quad then$$

$$Edge1.weight + Edge\ 2.weight$$

$$= Weight\ object, \quad if\ Edge1.NumberOfTransfers = 1$$

$$= Weight\ object\ with\ the\ number\ of\ transfers\ in\ the\ journey\ incremented\ by\ 1, otherwise$$

Comparison of Weights

The implementation of Weight keeps a track of different criteria that are important to the user. However, priority queue in Dijkstra's algorithm needs a way to compare two different weights. The criteria in the 'Weight' belong to different dimensions. It includes the total time of the journey, the total distance travelled, total walking distance and even number of transfers involved in the journey. Hence, these different criteria are converted to a single dimension so that they can be easily compared. They are converted to 'beads of sweat,' which symbolizes the effort on part of the user.

If given a choice, most people prefer to transfer onto another bus in the middle of the journey only if they save time significantly. Taking a bus transfer can be complicated for some people. It increases the chances of making a mistake, especially if the passenger is new to the city or the area. The other bus could be running late, and that might potentially diminish the advantage of taking a transfer. For others it is simply inconvenient.

Moreover, most people prefer to ride one bus for longer, or wait at a bus stop rather than walk to a bus stop which is further away. Walking to another bus stop requires more effort, and it can get tricky to catch the bus if it is not running on schedule. If the user has to walk to bus stop further away then they expect a significantly shorter duration of journey.

Therefore, the following formula was devised.

$$Total\ Weight =$$
$$Total\ time\ of\ journey * 1\ bead\ of sweat/second$$
$$+ Number\ of\ transfers * 20 * 60\ beads\ of\ sweat/transfer$$
$$+ time\ spent\ walking * 2\ beads\ of\ sweat/second$$

According to this formula, travelling in one bus for 60 minutes is equivalent in effort to taking one transfer and saving 20 minutes. So if a user is expected to make one transfer then the person should save more than 20 minutes in the total journey. Additionally, the effort that a user has to make in walking is considered three times the effort of spending the same amount of time traveling in a bus. The walking effort is tripled in the formula, because the total time spent walking is multiplied by two, and the time spent walking is already included in the total time of the journey.

It should be noted that additional criteria can be easily added to the formula in a similar fashion, e.g. cost of the journey. Moreover, some of these considerations can be very specific to individuals and situations. In order to accommodate for them, these values can be entered by the user. The Dijkstra's algorithm is solved when the user makes a request, and introducing the values entered by the user in this formula before solving Dijkstra's would not be relatively easy.

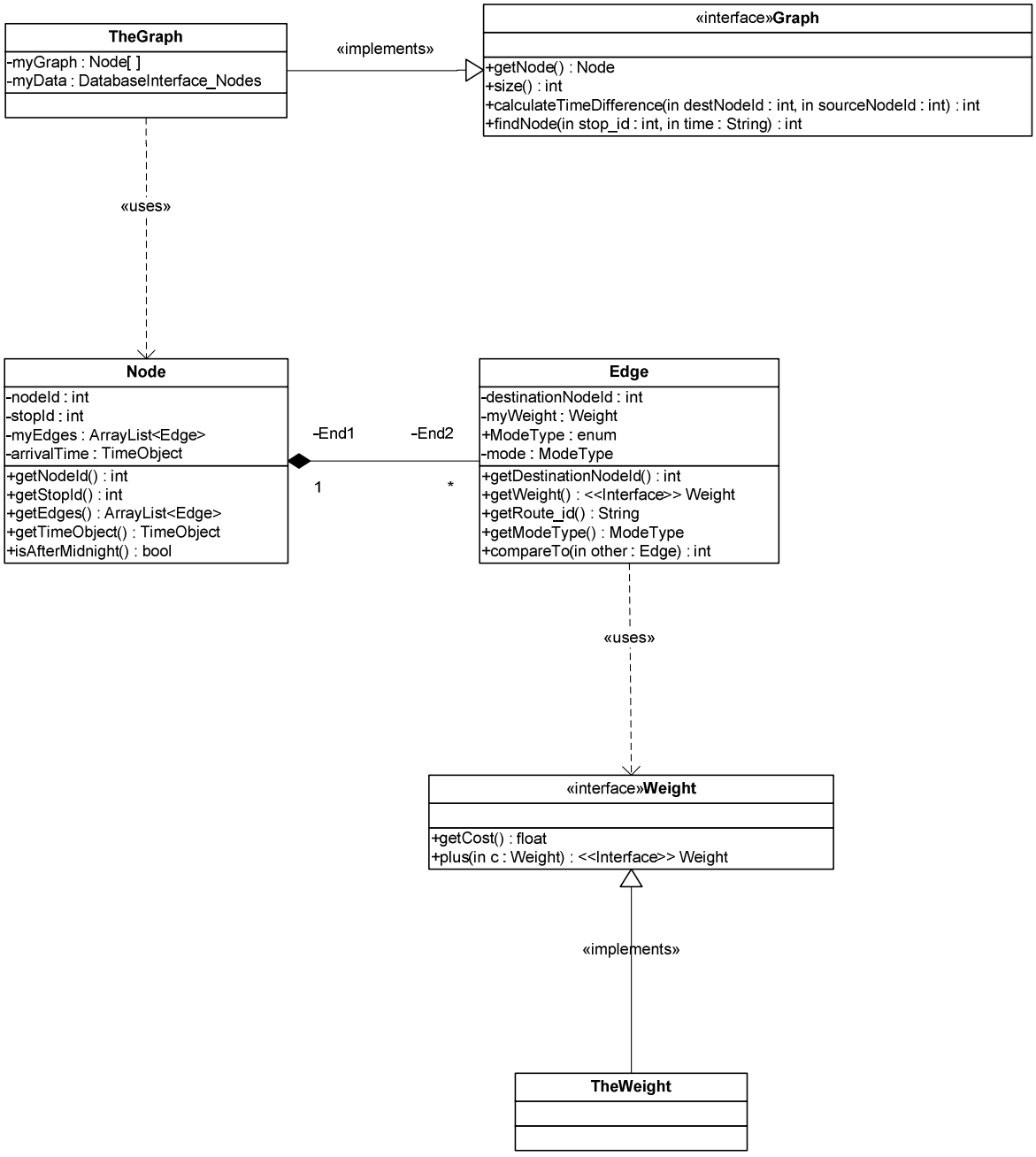


Figure 21: UML diagram for Dijkstra's Graph

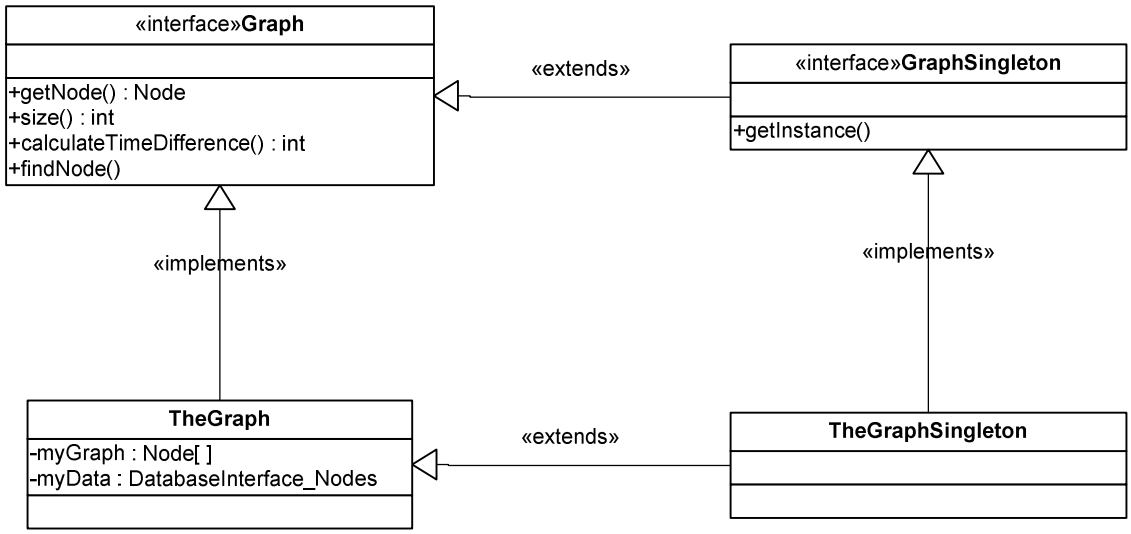


Figure 22: Implementation of Singleton Pattern for Dijkstra's Graph

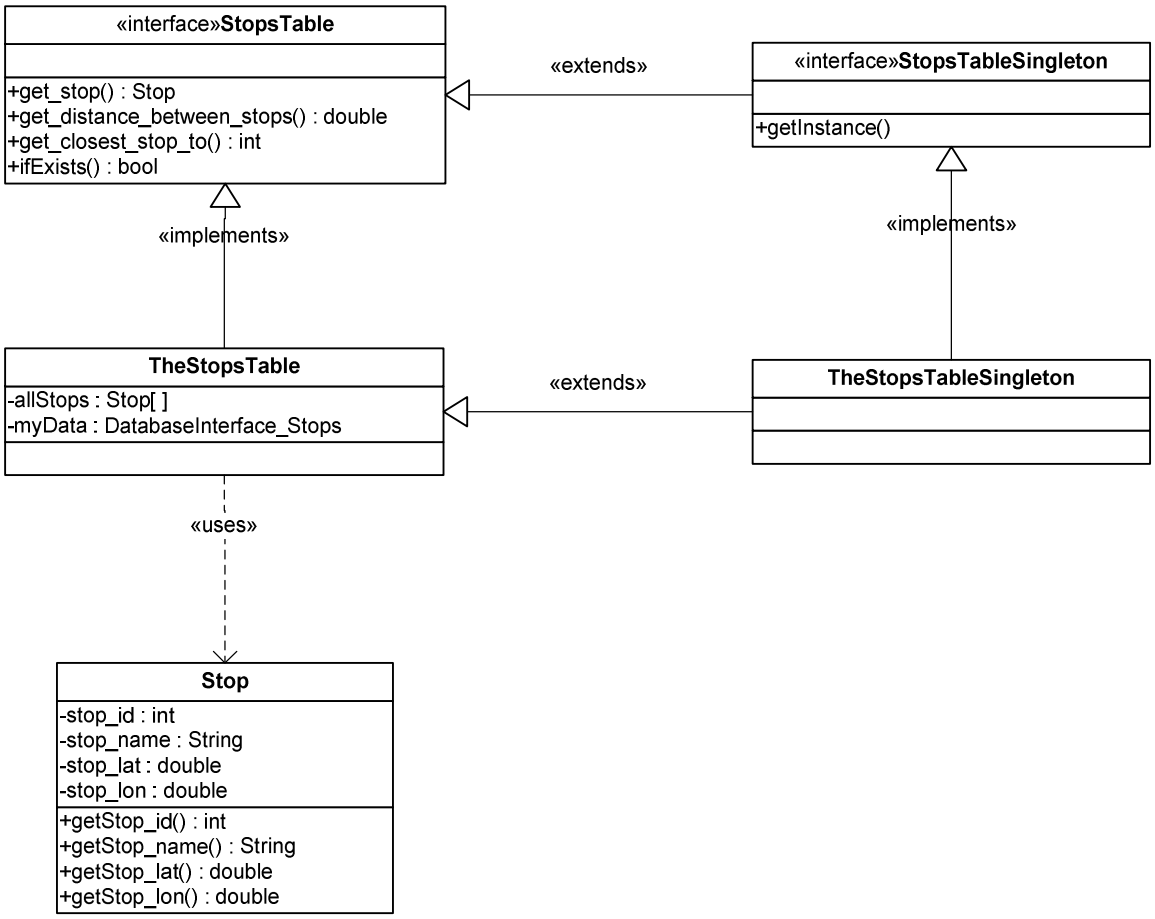


Figure 23: Implementation of Singleton Pattern for StopsTable

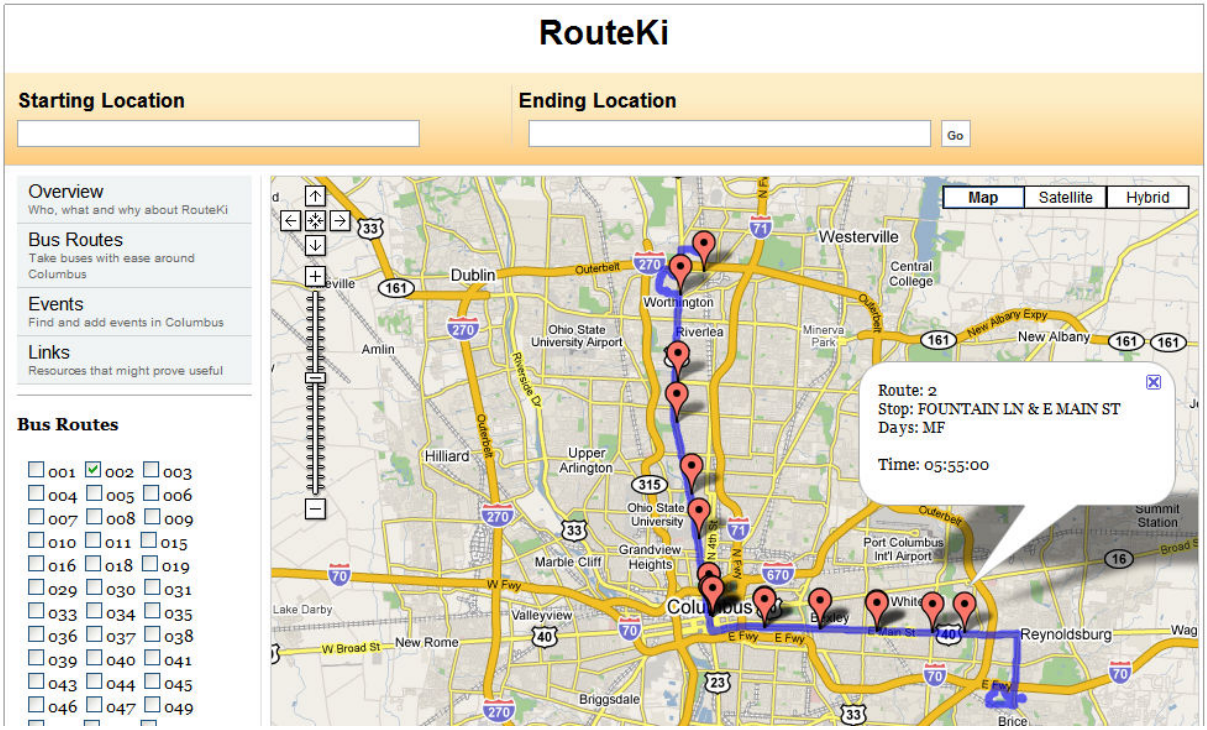


Figure 24: User Interface

CHAPTER 6

Test

White box testing approach was used in the project. Junit unit tests were written for different classes and complex methods. These tests consisted of representative cases, trivial cases and boundary conditions. These tests helped to find bugs in the existing code, and when the code was modified. Code was committed to the repository only after all regression tests were completed successfully. Lastly, system tests were created to see test specific functionality of the program and to ensure that the program works as a whole. Junit tests verified that all the edges added to the graph met different conditions, that the internal representation of the graph was accurate, and that the graph met the boundary conditions. Tests were written to specifically test the implementation of the 'Weight' interface, and if the walking edges met all the conditions mentioned previously in Chapter 6. Below is a brief description of the tests were used to verify that the Dijkstra's algorithm correctly created the shortest path tree.

1. Test to see that if T_k is part of the Dijsktra's shortest path tree, then the shortest path to T_k was found.

For every node T_j in tree

For every node T_k in tree and adjacent to T_j

assert (Weight(source, T_k) <= Weight(source, T_j) + Weight(T_j, T_k))

2. Nodes that did not get reached are farther away from the destination.

For every node T_j in tree

For every node T_k not reached and adjacent to T_j

assert (Weight(source, destination) <= Weight(source, T_j) + Weight(T_j, T_k))

CHAPTER 7

Future Steps

Mobile Application

Functionality of the application will be useful in a mobile phone. A mobile application will help students find activities of interest even when they are not in front of their computer terminal. Nowadays popular mobile phones also have global positioning systems (GPS) on them. The user will be able to find a route useful the application from his/her current location to a location of interest on campus. It will help new students explore the campus without fear of getting lost. Moreover, the GPS functionality can be capitalized to enhance the GIS wiki aspect of the application.

Building an Online Community

Successful adoption of the application (the web and mobile application) depends on how many people on campus adopt and use it on a regular basis. It is important to get students excited about its concept. One of the reasons of conducting a survey of student organizations was to gauge their interest in such a concept, and to get a better idea of their needs. After its release, it will be important to listen to feedback of users and implement changes appropriately.

In order to develop a critical mass of virtual users it will be important to network with a larger number of people. Publicizing the application during the orientation of freshman and international students would help students get networking benefits out of the application in the initial period of their time in college. Such students will also tend to use the application for the rest of their time in college, and help spread the word of mouth. Networking with the Office of Information Technology, Office of Student Life, Undergraduate Admissions and First Year Experience, Office of International Affairs and Student organizations would help in developing credibility, reaching a wider audience, and getting good ideas.

Geographic Information Systems Wiki

Giving the ability to users to add their college buildings on the map, inform others about road blocks, new constructions, and adding information of public transit systems directly to the website would be innovative ways of helping other students on campus. Such features would also encourage students who study in other colleges, but live in Columbus to use the application.

CHAPTER 8

Conclusion

In this chapter, the discussion on the dynamic user generated content and the multi-modal routing algorithm for public transportation has been summarized. In Chapter 2, the need for an application that helps students recognize activities of interest on campus out of the thousands of activities has been outlined. The Ohio State University is one of the largest campuses in the United States. Over 61,000 students from over 100 countries enrolled in Autumn Quarter 2008 (Statistical Summary, 2009). The university owns over 900 buildings and 15,910 acres of land (Statistical Summary, 2009). Schedules of public transportation agencies can be confusing. An application that will help students find activities of interest and help them find a route to the location will help students feel more comfortable and encourage them to proactively seek opportunities of growth. A survey sent out to all student organizations on campus helped to indentify features of the application that would be beneficial for the end users. Not a single online tool/website met all the needs of the student organizations. In Chapter 2, the requirements work products for such an application have been described. In Chapter 3, the database that is required for the algorithm has been explained. The data is obtained from public transportation agencies in the format of Google Transit Feed Specification.

This data is converted into a MySQL database. This database is then used to construct the Dijkstra's graph in the memory. In Chapter 4, Dijkstra's algorithm has been explained with the help of a priority queue implementation. The running time of such an implementation of Dijkstra's algorithm is ($n \log n + m \log n$).

In Chapter 5, the application of Dijkstra's algorithm for bus routing has been explained. Each node in the graph represents latitude and longitude of a bus stop, along with the time when a bus arrives at the bus stop. Hence, the graph for public transportation is three dimensional. Latitude, longitude and time form the three dimensions. Three different kinds of edges connect the nodes in the graph. They include a bus edge, waiting edge and walking edge. If a bus route exists between two nodes then the nodes are connected by a bus edge. Waiting edge connects two nodes that belong to the same bus stop, whereas a walking edge connects two nodes that are in close enough in space-time such that a passenger can walk from one node to the other. Naïve implementation of adding walking edges to the graph is a very expensive process, with respect to total time taken by the process. A more efficient way of adding walking edges was explained and its running time analyzed. The running time of this algorithm is $\theta\left(n * \frac{s}{k}\right)$, where n is the number of nodes in the graph, s is the number of bus stops and k depends on the MAX_WALKING_DISTANCE. As the value of MAX_WALKING_DISTANCE is increased the value of 1/k becomes large and the algorithm takes a long time. However, if the MAX_WALKING_DISTANCE is less than or equal to 240 meters then the walking edges can be added to the graph in about 100 seconds for the entire city of Columbus.

The three different types of edges are associated with different weights. A bus edge is usually preferred over a walking or waiting edge. However, walking or waiting edge are chosen is there is a significant reduction in the total time of the journey. The weight of a route in the

growing spanning tree is calculated on the amount of 'effort' required by the user. Effort is calculated by a combination of factors including the total time of the journey, the amount of walking distance involved and the number of buses changed in the journey. Chapter 8 briefly mentions how the implementation of Dijkstra's algorithm was tested.

In Chapter 7, future steps of the application have been explored. The ability for students to find interesting events and find a route to them would be useful in a mobile device. In addition, a critical mass of users would be required to generate data on a regular basis. Hence, it would be critical to build an online community by promoting it strategically in the real world. Networking with organizations and getting support of different offices of the university could help towards that end. Lastly, using the experience of managing and motivating users to generate online content can be capitalized to develop a GIS wiki to benefit countries that lack digital map content.

Bibliography

Activities & Organizations. (2009, March). Retrieved March 24, 2009, from The Ohio Union: http://ohiounion.osu.edu/studentorgs/orgs_directory.asp

Cormen, T. (2003). *Introduction to Algorithms.* New York: McGraw-Hill.

Dijkstra's Algorithm. (2009, March 23). Retrieved April 12, 2009, from University of Saskatchewan: http://www.cs.usask.ca/classes/371/projects/graphs/tutorial/advanced/dijkstra/dijkstra.html

Google. (2009, February 26). *Google Transit Feed Specification*. Retrieved March 25, 2009, from Google Code: http://code.google.com/transit/spec/transit_feed_specification.html

Meridian World Data. (2009, April). *Distance Calculation*. Retrieved April 13, 2009, from Meridian World Data: http://www.meridianworlddata.com/Distance-Calculation.asp

Office of International Affairs. (2007-08). *Annual Report.* Columbus: The Ohio State University.

Skiena, S. (n.d.). *Dijkstra's Algorithm*. Retrieved March 24, 2009, from Computational Graph Theory with Combinatorics : http://www.cs.sunysb.edu/~skiena/combinatorica/animations/dijkstra.html

Statistical Summary. (2009, March). Retrieved March 24, 2009, from The Ohio State University: http://www.osu.edu/osutoday/stuinfo.php