DATE: January 31, 2007

TO: CheolHee Park

FROM: Yoni Ben-Meshulam, Garrett Cooper, Derrick Huhn, Patrick Lowry

SUBJECT: Rideshare Web Application Proposal

INTRODUCTION

Rideshare is a web application connecting drivers and passengers who seek to participate in carpools. It will do so in an efficient manner by minimizing travel time and distance, while taking into account the preferences of customers.

These days, fuel costs are pressuring people to leave their cars at home and seek alternatives. One of the best choices is cooperative transportation, or carpooling. The carpooling alternative is one which is widely recognized as a good solution for alleviating traffic congestion [5][6][9]. Currently, there is very little being done to assist people in organizing carpools. Websites that are dedicated to carpooling do little more than offer their users a bulletin-board for posting their ride offers and requests. Users are left to search for and organize carpools themselves [7][8]. We want to develop a web service to fill this void, and automatically match users to one another based on compatibility and all-around efficiency.

The RideShare application will help people coordinate rideshare programs and eliminate the need to plan an ideal route. People who regularly share rides will significantly reduce their ecological footprint and save on travel expenses. In addition, our solution will help people find rides that would otherwise not be possible without exhaustive effort.

Our team of four has diverse backgrounds. Patrick and Derrick are both receiving their BSs in Computer Engineering, and have extensive experience in web development and the Linux/Apache/MySQL/PHP (LAMP) platform. Additionally, Derrick has experience with Google Maps, and Patrick has previously implemented a few evolutionary algorithms. Garrett and Yoni - both inexperienced in LAMP development - have worked on projects in PERL, a scripting language similar to PHP. Yoni is receiving his degrees in Computer Engineering and Mathematics and has experience in artificial neural networks, and evolutionary algorithms. He has also been involved with the Rideshare optimization problem for the past year. Garrett is currently earning his BS in Electrical Engineering, with an emphasis on computer design.

In the following sections, we discuss the underlying computational problem involved in developing the rideshare model. We present our approach for solving this problem efficiently and effectively, while providing a friendly and intuitive interface to the user. We will describe the development framework and task division; both essential to the success of the Rideshare application. We will then break down the process of developing the application in terms of the tools and resources we will need. Finally, we introduce a schedule for completing the tasks, and divide the work into four major modules. We conclude with a summary of our main points, and a positive outlook toward the development of Rideshare.

THE RIDESHARE CHALLENGE

Our web application must solve the problem of taking a dynamic set of ride offers and requests, and finding efficient routing solutions. In other words, we must maximize the number of rideshares, while adhering to the user preferences, and route and time constraints. We must also provide a user-friendly environment for managing profiles, entering rideshare requests or offers, and viewing results. This environment must be simple and intuitive for it to be useable by and useful to all users.

In order to solve the problem of matching drivers and riders, we must understand the fundamental problem with which we are dealing. In academia and industry, this problem is categorized as an optimization problem. We must take the set of drivers (D) and riders (R), and find routes between their starting positions and destinations such that we maximize the number of riders served, while taking into account user preferences and constraints, route length, and the user-specified time-windows.

A similar problem, which is called the Vehicle Routing Problem, is stated as finding minimal routes for a certain number of vehicles (k) to visit a certain number of customers (n), where the vehicles start at a central location called a 'depot.' Our problem, though similar, is different in that we allow for some customers to not be matched in the case that their criteria is not met, or if they are too far out of the way to route a vehicle to them. In addition, our problem has several constraints, which can be summarized as follows:

Constraints:

- Each route must originate at the drivers origin and terminate at their destination.
- Each vehicle's schedule must adhere to the driver and riders' time window constraints.
- Users will be required to specify definite departure and/or arrival times.
- The total number of riders in a vehicle at any given time cannot exceed that vehicle's capacity.
- User preferences, such as gender, ratings, and age, must be met.

Objectives:

- Maximize the number of riders matched.
- Maximize the total efficiency of the routes, in terms of time and distance, on a global level.

This problem is formally known as the Capacitated Multi Depot Vehicle Routing Problem with Time Windows (CMDVRPTW). It is a variation on several well-known problems, which have been shown to be NP-Complete, meaning that the time it takes to solve them grows exponentially relative to the number of inputs. Timeliness of results is important, since a result that isn't available in a timely manner is useless. This exponential growth means that, as the set of inputs grows large, algorithms that solve this type of problem can potentially take longer to complete than the history of the universe. This last requirement necessitates the use of an algorithm which does not run in exponential-time, rather completes in some bounded time-frame tolerable by humans. [1][2][3][4]

Thus, our problem cannot be practically solved directly, i.e. the optimal solution would take too long to find. Fortunately, several methods have been developed for finding approximately optimal solutions. The field of Operations Research deals directly with this problem. We will use several of the approaches used by academics for optimizing similar problems, and tailor their methods to our variation of the problem (CMDVRPTW). [1][2][3][4]

One of the most important aspects of our project will be providing an intuitive and user-friendly interface to allow the integration of the service into people's daily lives. If the interface is confusing, tedious, or annoying, users will choose not to use it. We must ensure that each user is able to use the tools which we provide them with to create rideshares. Several map-based interface technologies have recently emerged and could potentially facilitate an intuitive interaction between Rideshare and users.

OUR APPROACH TO IMPLEMENTING RIDESHARE

Many of the current ride matching services act only as a forum for riders and drivers to post offers or requests [7][8]. We will eliminate the need for users to search through forum posts by matching them according to preferences. RideShare will provide a transparent interface which users can submit their desired itinerary and preferences for a trip in the hopes to be matched with a driver with a similar itinerary.

In order to implement the application, we plan on building a web-based interface framework with a database on the backend. Multiple packages are available at each level: the hosting platform (Windows, Linux), the web server (Apache, IIS, Tomcat, lighttpd), the database server (MySQL, PostgreSQL, Oracle), and the web toolkit (Ruby on Rails, PHP, Perl, ASP). We have past experience with the combination (Linux, Apache, MySQL, PHP) codenamed LAMP, and as such already have a server with it available [25].

On this server, we will install the rideshare application, which will be comprised of four main modules: Control, User Interface, Optimizer, and Database, as shown in Figure 1.

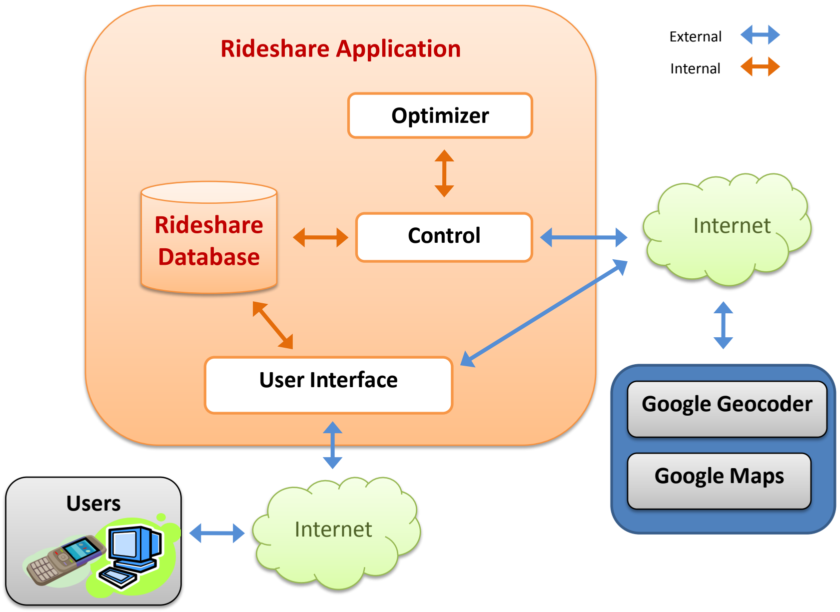

Figure 1: Rideshare Block Diagram

User Interface

Our approach for creating an intuitive and effective user-environment is to use a map-based interface, commonly known as a map "mashup." We plan on using the Google Maps Application Programming Interface (API) as the foundation for our interface. Users will have the option of inputting the addresses on a form, or selecting previous locations marked by icons on the map. The website will be a mix of map-based and form-based interactions, with an emphasis on minimizing forms. Figure 2 shows an example of what a ride entry page might look like.

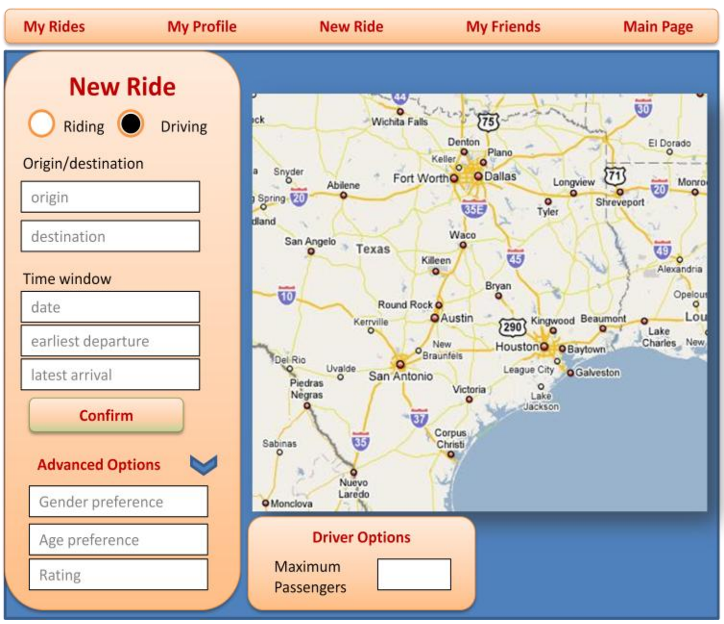

Figure 2: Sample User Interface

Optimizer Module

After users have submitted requests, the Optimizer (Figure 3) will match drivers with riders according to their desired origin and destination, available time windows, and preferences. These preferences will most likely include user ratings, gender, and age.

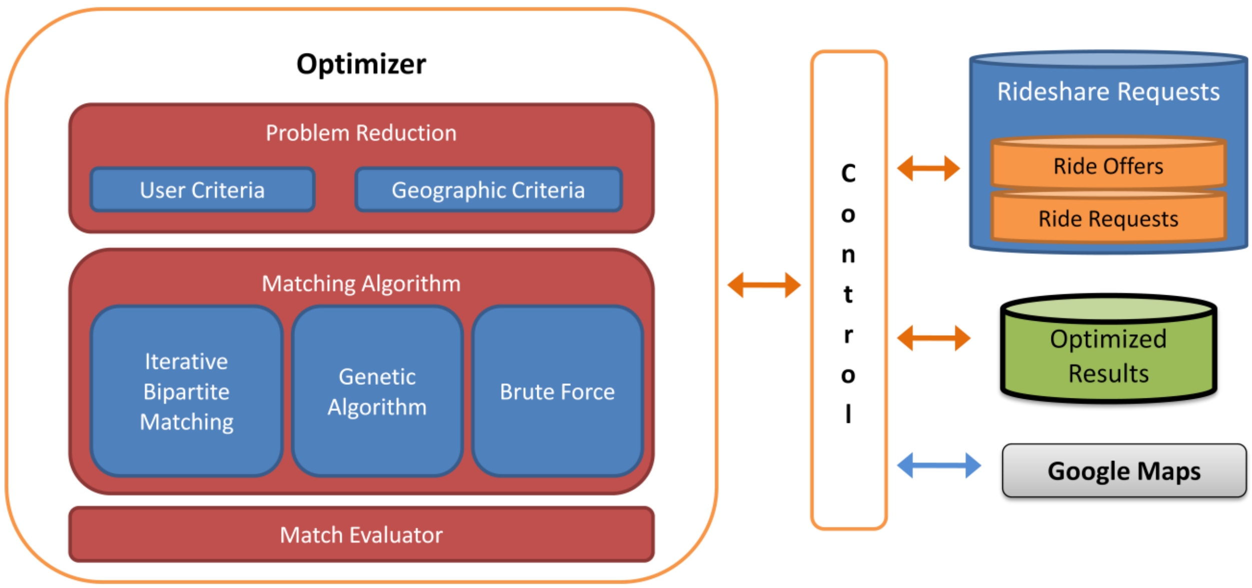

Figure 3: Optimizer Block Diagram

In order to find an optimal solution, we first need to reduce the search space of the problem at hand. Search space reduction, or pruning, is the process of eliminating possibilities which are clearly invalid, and thus reducing the amount of work necessary for finding matches. The Problem Reduction component of the Optimizer module will reduce the number of possible solutions by excluding matches between a driver and riders that are outside their route and time windows or do not share similar preferences. Using this reduced set, the optimizer algorithms will then sift through possible solutions looking for the most efficient routes. The Optimizer module will be the crux of the RideShare project, so a significant amount of the design process will be dedicated to researching and implementing usable solutions.

Optimization Algorithms

We will implement a variety of algorithms proven to be capable of approximating NP-complete problems effectively and efficiently. We will implement our solutions using three algorithmic approaches: simple-iterative (commonly referred to as brute-force), iterative bi-partite solver, and a genetic algorithm. We feel that implementing these algorithms and analyzing their results is necessary to provide users with routes that are efficient and adhere to user preferences. In addition, we will develop a real-time visualization of these algorithms to help us visually understand and analyze the results produced by our algorithms.

The brute force method will simply iterate over all possible combinations of rides, i.e. the "naive" approach to finding the optimal solution. This method is meant to be used for comparing the other implementations, and for presenting the complexity of the problem. It will most likely not actually be used by the application for optimizing results of more than a very small number of users.

One way to optimize over the routes is to utilize the fact that our problem can be viewed as a mathematical construct called a *bipartite graph*. This method greatly reduces the complexity of solving the optimization problem, but allows for pairing only one rider per driver. Viewing our problem as a *maximal matching problem* on a bipartite graph allows us to employ several algorithms which are dedicated to this and similar problems. Since bipartite graph matching allows only one rider per driver, we will perform bipartite matching iteratively, adding one passenger per driver at a time. For detailed information on graph theory and maximal matching, please refer to Appendix B: Graph Theory.

The *genetic algorithm* takes its approach from nature, mimicking the process of natural selection based on fitness. The genetic algorithm takes a set of randomly generated solutions and ranks them according to a fitness function (also known as a scoring function). It then takes the best few candidates and mutates them (usually some small permutation), then ranks those results and mutates the best of those, etc. Some implementations have it also occasionally pull in a completely new random set to avoid dead-end paths on the road to the best solution. [26]

Control Module

The Controller module (Figure 4) of Rideshare will work in concert with the Optimizer module, acting as an interface to the database that holds all ride requests and offers. We have restricted access to the database through the Controller in order to reduce code duplication in all the modules. For example, if all components of the Optimizer module contained separate code blocks that all had the same database functionality, maintaining the code could be problematic if changes were made to the database. Defining access in one module reduces the complexity of code maintenance and provides a uniform API for developers interfacing with the database.

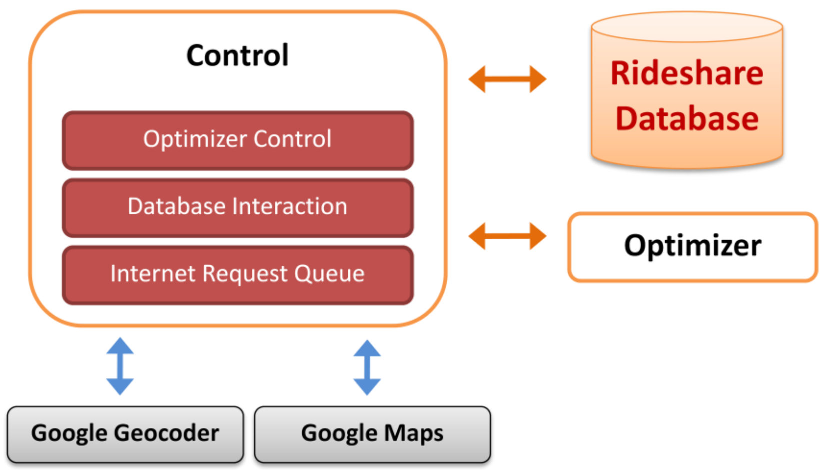

Figure 4: Control Block Diagram

The Control module will also be the interface for all modules to use when querying routing data on Google Maps.

Results

After drivers and riders have been matched, they will be notified via e-mail or text message, depending on their communication preferences. The routes generated will be available on a Google Maps canvas and an itinerary will be created for users to download. Once results are ready, a user will be able to review their possible matches, and confirm or deny a Rideshare. In order for a Rideshare to be established, both parties must confirm the match. This is referred to as a "handshaking policy."

RIDESHARE DEVELOPER ACTIVITIES AND RESPONSIBILITIES

In order to create a development environment, we have obtained and set up a server, and installed the various programs and tools we will need on it. The UT Communications Society loaned us a computer, as facilitated by Patrick and Derrick. After that, Derrick installed LAMP. Since then, Patrick and Garrett began constructing the website's front end in PHP. Garrett and Yoni do not have as much experience in PHP as Patrick or Derrick, so as work on the user interface continues, Patrick will help them gain expertise in the language.

During this time, Derrick began installing a version control and content management system (CMS). We decided on an open source CMS called Trac, which includes Subversion (SVN), a Wiki, and Bug Tracking system [24]. Simultaneously, Yoni has been researching the problem definition and several algorithms and approaches for solving it [10].

During the coming week, we will determine the interfaces between the User Interface, Database, Controller, and Optimizer modules. These interfaces represent how the different modules "talk" to one another. Once they are established, the tasks of developing each module can be performed in parallel. All team members will be involved in the process of determining module interfaces.

The first features of the User Interface module will be those related to the database support for user profile management. These will be developed by Patrick and Garrett. Yoni and Patrick will work on support for ride request entry and accessing optimization results. Simultaneously, Derrick will be working on Google Map querying in the Controller.

Once tasks these are complete, we all will work on the front end of the user interface: Derrick and Yoni will collaborate on the map-based toolkit, and Patrick and Garrett will work on the site map and web page templates. By the end of February, we should have most of the web application functioning and ready for testing.

Throughout the first stages of the project, we will still be developing our approach for the optimization problem. Once most of the back end is ready, we will begin writing our optimization functions, while Derrick writes the Control module. Yoni will design a method to reduce the "search space" by eliminating obviously incompatible matches.

The actual optimization algorithms will be developed by all team members since they are a significant portion of the total work. Garrett and Patrick will work on the genetic algorithm, and Yoni and Derrick on the brute force and bipartite matching algorithms.

Throughout the development process, each team member will test their functions on a local, module-level. Garrett will be in charge of writing a testing framework for the web application as a whole. Towards the end of February, we hope to have tested much of the functionality of the website, and we will use March to develop and test the optimization algorithms.

Once these are done, we will have all the basic functionality in place, so we can test the program in its entirety. The rest of semester we will work on the advanced features, as well as refining features we have already implemented. The last few weeks of April, we can work on supplemental documentation, usability, and visual design.

We have divided and timed these tasks according to the complexity, amount of effort, skills, and points of interest of each team member. Each major milestone has an "owner." As the semester unfolds, we will tweak this task division, with each milestone owner reporting on the progress made, and restructuring the task division as necessary to complete everything on time.

Please see the Gantt chart in Appendix A for a detailed description of task divisions and milestone dates.

RESOURCES

Dr. Constantine Caramanis is our faculty sponsor. His research interests include optimization, including combinatorial optimization problems like our own. CheolHee Park is our technical TA. He is currently a student of the WNCG here at UT, working on his Ph.D. His research areas include wireless communications and networking. Also, he is skilled in algorithm analysis and he is familiar with C++.

We also have access to several academic papers and websites to reference. Google Scholar and IEEE Xplore are both search engines designed specifically for searching academic papers [12][13]. These will prove particularly useful for algorithm research, though if we need to look for more specific papers, we can browse through The VRP Web, which is devoted to the vehicle routing problem [4]. To a lesser extent, we can use Wikipedia to search for general information on any algorithms or tools we may use [15].

Not all team members will be versed in the programming languages or software tools we will be using. For PHP, we can reference their official website, which references useful documentation and frequently asked questions [16]. The MySQL developer's page has support from the open source community so documentation and help is freely available[17]. If we need help with the Google Maps API, we will use their documentation in addition to their developer forums. In addition, the UT Library offers several online resources, including programming manuals.

MATERIALS

The web server we are using was loaned to us by the UT Communication Society [19]. Luckily, it was loaned to us as a dedicated server, which means we were able to completely change its software down to the operating system, and it is available to us on a 24/7 schedule. It is an older computer, with only a 500MHz Pentium 3 processor, 512KB of L2 cache and 128MB of RAM. The server is connected to the internet through a 10/100 cable to the UT network in ENS, and from there to the internet on a T3 line, which is very fast. We are able to access the server through a secure connection from any computer with internet access. Fortunately, all the software (PHP, MySQL, Ubuntu, Apache, and Trac) we are likely to use is freely available and open source [16][17][20][21][22]. We will also be using a free software service as we interface through the Google Maps API [23].

We will take several measures to control costs. Primarily, the Communications Society loaning us a server has drastically cut costs. Additionally, we have limited the scope of the problem to what we feel can be completed by the April demo date. We will focus on completing the features set out in this document and will not add anything extra until we have completed them. The only costs that could be incurred are our time and the running time power costs of the server, which is defrayed by the university. To get a feel of our time budgeting, please reference the Gantt chart in Appendix A.

DEVELOPMENT MILESTONES

1. Obtain Server (Jan 18)
2. Install LAMP and Trac (Jan 25)

   Install and setup Ubuntu Server, MySQL, PHP, Apache2. Configure Subversion and Trac. Create accounts and setup development infrastructure
3. Overall design of interfaces between the main modules (February 1st, all team members)

   Database, Controller, Optimizer, User Interface
4. Framework (February 8)

   Need functionality for creating and accessing user profile management, including personal info, blacklist, ride requests, and ride offers. (Patrick and Garrett)

   Database tables for interfaces, including ride requests and results (Yoni and Patrick), Google Maps queries. (Derrick)

   Controller to database communication link (Yoni and Patrick).
5. Establish our algorithmic approach. (All Members)
6. Finish Backend for UI/Optimization (February 29)

   User Notification (email, text, rss) (Derrick)
7. Front end for UI (February 29)

   Google maps visualization: (February 22, Yoni and Derrick)

   All map-based interactivity (February 29, Yoni and Derrick)

   Site map and page templates (February 29, Garrett and Patrick)
8. Testing

   Generating test-cases (February 29, Garrett)

   Algorithm performance measures, visualization (March 21, Garrett)
9. Optimizer (March 21)

   Control Module, Problem Reduction

   Optimization algorithms:

   - Genetic algorithm (March 14, Garrett, Patrick)
   - Brute Force (March 7, Yoni)
   - Bipartite Graph Matching (March 14, Derrick)
   - Iterative Bipartite Graph Matching (March 21, Derrick)

10. Optional features

    User profile management: friends, history of user requests/routes

    Facebook Integration, RSS Feeds

11. Refinement & Testing (April)

EE464 MILESTONES

1. Jan 31: Proposal Due
2. Design Review Due (February 14)
3. Intellectual Property Report due (Feb 28)
4. Pre-Demo Performance milestone with TA (Feb 28)
5. Progress Report Due (March 20)
6. Final Project Demo Milestone with TA (April 1)
7. Oral Final Presentations I (April 29)
8. Final Report Due (May 1)
9. Oral Final Presentation II (May 1)

CONCLUSION

In this report, we have presented our idea for a novel and promising web application which facilitates carpooling. We believe that our product will be a friendly way of making carpooling much easier to coordinate.

The underlying problem we are solving is categorized as NP-Complete. This may be the biggest challenge of the project, and so we will implement several methods for performing the optimization, and assess the performance of each.

Our schedule will be pretty tight, with two months to develop the application. We have broken down the development tasks into four modules which are minimally inter-dependent, and assigned them to team members according to their skills and interests.

The resources necessary for completing this project are mostly freely available, so we won't have to worry about incurring costs.

We look forward to developing Rideshare during the next semester, and have faith in our ability to deliver the project on time.

Include (not in this order…):
Purpose of the report
Impace of time and resource limitations on the project
Realistic assessment of the outcome
The technical significance of your solution to the problem
Importance of the project
touch on the reasoning discussed in the first paragraph.
Briefly mention the modules (don't go into any detail) and explain why what we are doing is important
Team and tasks
Optimistic outlook and excitement!

Be as upbeat as you can!

REFERENCES

[1] Paul E. Black, "Vehicle Routing Problem", *Dictionary of Algorithms and Data Structures,* [Online Document], Paul E. Black, ed., U.S. National Institute of Standards and Technology. 17 December 2004. [Cited 2008 January 26] Available: http://www.nist.gov/dads/HTML/vehicleRouting.html

[2] Wikipedia contributors, "Vehicle Routing Problem," *Wikipedia, the Free Encyclopedia,* [Online Document], December 16, 2007, [Cited 2008 January 27], Available: http://en.wikipedia.org/w/index.php?title=Vehicle_routing_problem&oldid=178307741

[3] Monaldo Mastrolilli, "Vehicle Routing Problems" [Online Document], 2001 June 8, [Cited 2008 January 28], Available: http://www.idsia.ch/~monaldo/vrp.html

[4] Bernabé Dorronsoro Díaz, "The VRP Web" [Online Document], March 2007, [Cited 2008 January 29], Available: http://neo.lcc.uma.es/radi-aeb/WebVRP/

[5] Washington Department of Transportation, "Washington State Freeway HOV System", [Online Document], 2008, [Cited 2008 January 27], Available: http://www.wsdot.wa.gov/hov/

[6] North Carolina Department of Transportation, "HOV Planning Guidelines", [Online Document], 2008, [Cited 2008 January 27], Available: http://www.ncdot.org/projects/hov/hPlaGuidelines.html

[7] "E-Rideshare" [Online document], 2005, [Cited 2008 January 27], Available: http://www.erideshare.com/

[8] "CarpoolConnect" [Online document], 2008, [Cited 2008 January 27] Available: http://www.carpoolconnect.com/

[9] Wikipedia Contributors, "High-occupancy Vehicle Lane," *Wikipedia, the Free Encyclopedia*, [Online Document], 2008 January 24, [Cited 2008 January 27], Available: http://en.wikipedia.org/w/index.php?title=High-occupancy_vehicle_lane&oldid=186485776

[10] Yoni Ben-Meshulam, "Optimization," 2008 January 27, [Cited 2008 January 27], Available: http://128.83.50.221/trac/rideshare/wiki/Optimization

[11] Constantine Caramanis, "Constantine Caramanis," [Online Document], 2008, [Cited 2008 January 29], Available: http://users.ece.utexas.edu/~cmcaram/

[12] "Google Scholar," [Online Document], 2008, [Cited 2008 January 29], Available: http://scholar.google.com/

[13] "IEEE Xplore," [Online Document], 2007, [Cited 2008 January 29], Available: http://ieeexplore.ieee.org/Xplore/dynhome.jsp
[15] Wikipedia Contributors, "Wikipedia," *Wikipedia, the Free Encyclopedia*, [Online

Document], 2008 January 29, [Cited 2008 January 29], Available: http://en.wikipedia.org/wiki/Main_Page

[16] "PHP: Hypertext Preprocessor," [Online Document], 2008, [Cited 2008 January 29], Available: http://www.php.net/

[17] "MySQL Developer Zone," [Online Document], 2008, [Cited 2008 January 29], Available: http://dev.mysql.com/

[18] "Google Code," [Online Document], 2008, [Cited 2008 January 29], Available: http://code.google.com/

[19] "UTComSoc," [Online Document], 2008, [Cited 2008 January 29], Available: http://utcomsoc.org/

[20] "Ubuntu," [Online Document], 2008, [Cited 2008 January 29], Available: http://www.ubuntu.com/

[21] "The Apache HTTP Server Project," [Online Document], 2008, [Cited 2008 January 29], Available: http://httpd.apache.org/

[22] "The Trac Project," [Online Document], 2008, [Cited 2008 January 29], Available: http://trac.edgewall.org/

[23] "Google Maps API," [Online Document], 2008, [Cited 2008 January 29], Available: http://code.google.com/apis/maps/index.html

[24] "Subversion," [Online Document], 2008, [Cited 2008 January 30], Available: http://subversion.tigris.org/

[25] Bruce Timberlake, "Building a LAMP Server," [Online Document], 2006 May 10, [Cited 2008 January 30], Available: http://lamphowto.com

[26] Wikipedia Contributors, "Genetic algorithm," *Wikipedia, The Free Encyclopedia*, [Online Document], 2008 January 16, [Cited 2008 January 30] Available: http://en.wikipedia.org/w/index.php?title=Genetic_algorithm&oldid=184647559

APPENDIX A: RIDESHARE DEVELOPMENT SCHEDULE

Task	Owner	Date of Completion	Duration (Weeks)
Obtain Server	**Derrick, Patrick**	18-Jan	1
Install LAMP & Trac	**Derrick**	25-Jan	2
Install & Setup LAMP			1.5
Configure SVN, Trac, Wiki			1.5
Module-Module Interface Design	All	1-Feb	2
User Interface			0.5
Database			1
Controller + Optimizer			1
Framework	**Derrick, Patrick**		2.5
User Profile Management	Patrick, Garrett	8-Feb	2.5
Personal Info			1
Blacklist			1
Ride Requests (Drivers & Riders)			2
Interfacing Database Tables	Yoni, Patrick	8-Feb	2
Ride Requests			1
Results			1.5
Google Maps Queries	Derrick	8-Feb	2
Controller-Database Interaction	Yoni, Patrick	8-Feb	1
Backend for UI/Optimization	**Derrick**	29-Feb	3
User Notification			1
User Interface Frontend	**Patrick**	29-Feb	4
Google Maps Visualization	Yoni, Derrick	22-Feb	2
Map-Based Interactivity	Yoni, Derrick	29-Feb	2
Site Map & Page Templates	Garrett, Patrick	29-Feb	1
Optimizer	**Yoni**	21-Mar	8
Control Module	Yoni	21-Mar	2
Search Space Reduction Function	Yoni, Derrick	14-Mar	2
Optimization Algorithms	Yoni	21-Mar	8
Genetic	Garrett, Patrick	14-Mar	7
Brute Force	Yoni	7-Mar	5
Bipartite Graph Matching	Derrick	14-Mar	7
Iterative Bipartite Graph Matching	Derrick	21-Mar	7
Testing	**Garrett**	April	11
Test Case Generation	Garrett	29-Feb	2
Algorithm Scoring Function	Garrett	21-Mar	2
Algorithm Visualization	Garrett, Patrick	21-Mar	4
Testing	**Garrett**	April	11
Test Case Generation	Garrett	29-Feb	2
Algorithm Scoring Function	Garrett	21-Mar	2
Algorithm Visualization	Garrett, Patrick	21-Mar	4
EE464 Deliverables			
Proposal	Yoni, Garrett	31-Jan	n/a
Design Review	Derrick, Patrick	14-Feb	n/a
Design Review Oral Presentation	Yoni, Derrick	14-Feb	n/a
Intellectual Property Report	Yoni, Derrick	28-Feb	n/a
Pre-Demo Performance	All	28-Feb	n/a
Oral Progress Report	Garrett, Patrick	28-Feb	n/a
Final Demo	All	1-Apr	n/a
Oral Final Presentations I	All	29-Apr	n/a
Final Report	All	1-May	n/a
Oral Final Presentation II	All	1-May	n/a

January, 2008		February, 2008				March, 2008					April, 2008				
1/18	1/25	2/1	2/8	2/15	2/22	2/29	3/7	3/14	3/21	3/28	4/4	4/11	4/18	4/25	5/2

APPENDIX B: GRAPH THEORY

Some understanding of elementary graph theory is necessary in order to solve our optimization problem. We include this section to assist the reader in understanding the optimization problem at hand, and thus gain an understanding of the optimizer module's required functionalities. Mathematical definitions are marked with bullets.

- A *graph* is defined as a set of *vertices* $V=\{v_1,v_2,\ldots,v_n\}$ and a set of *edges* $E=\{e_1,e_2,\ldots,e_m\}$, for some positive integers m and n.
- The vertices can be thought of as nodes, and the edges as lines connecting those nodes, as shown in Figure A1

Our optimization problem can be viewed as a graph theory problem. In our case, the drivers (D) and riders (R) can be viewed as vertices, where two vertices are connected by an edge if they can be partners in a rideshare.

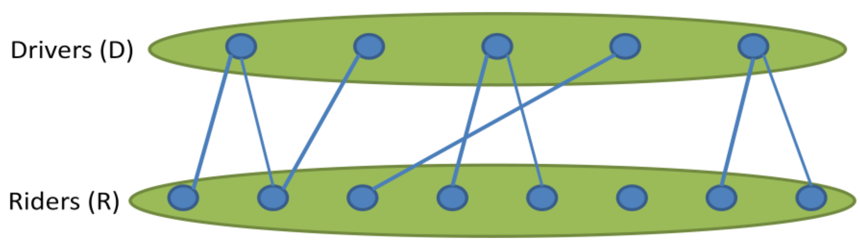

Figure A1: Bipartite graph of drivers and riders

- A *matching* is a set of vertices and edges. For a matching M, two vertices are said to be matched if M contains an edge which connects them.

Our optimization problem can be viewed as equivalent to the maximal matching problem. The maximal matching problem is the problem of finding a matching within a graph which includes the maximum number of vertices. Translated to rideshare, this means finding the most matches between drivers and riders, and thus providing rideshares for as many people as possible.

In particular, our optimization problem can be viewed as equivalent to a bipartite graph optimization problem.

- A *bipartite graph* is a graph in which the vertices of the graph can be divided into two disjoint sets, where no two vertices in the same set have an edge between them.

In our case, we have two sets of vertices, D (drivers) and R (riders), which can never have rideshares within the sets, i.e. no two drivers will be in a rideshare together, and the same holds for riders. This relationship can be seen in figure 7.3.1, where no edges are visible within the sets D and R.