The Campus Navigator

Bryan Boyd, Darla Haigler, Roger Pearce, Xinyu Tang, Akhil Patel, Nancy M. Amato
Parasol Laboratory, Department of Computer Science
Texas A&M University, College Station, Texas, 77843-3112, USA
{bcb2913, darlah, rpearce, xinyut, app6230, amato}@cs.tamu.edu

Abstract— **In this paper, we discuss our implementation of a web-based navigation system for the campus of Texas A&M University. This application represents the campus as a roadmap; a set of vertices and weighted edges. We designed a multilayer roadmap for our application that supports multiple methods of traversing each edge; in the campus environment, these methods represent different modes of transportation (e.g. walking, driving, riding bus). The resulting paths our navigator provides can be composed of segments of different transportation types; for instance, a long path across campus might include walking to a bus stop, riding the bus, then walking from the bus to the destination. An important benefit of our implementation is its simplicity. When dealing with a campus as large as Texas A&M, it is important that students, faculty, staff, and visitors be able to easily enter queries with little/no prior knowledge of the campus, and then clearly understand how to follow the resulting path to their destination. Also, an intuitive interface provides an adminstrator to easily expand/maintain the campus roadmap. Our efforts produced a navigation application that will aid the tens of thousands of people in finding their way around the campus of Texas A&M University.**

I. Introduction

Every year approximately 50,000 students attend Texas A&M. A substantial percentage of these students are newcomers to the campus and need a clear and intuitive way to find directions to locations such as buildings, classes, and eating establishments. Even experienced students often need directions to seldom-visited buildings. Ideally, these directions should be provided to students according to a chosen transportation method. For example, the path of a student driving across campus will differ greatly from the path of a student walking across campus. Our Campus Navigator application provides a simple, user-friendly interface to find directions around campus for a variety of transportation types.

Issues. There already exist several commercial navigation applications (e.g. Google Maps, Yahoo Maps, Mapquest) that provide users with directions from one place to another. However, these applications must search along existing roads; they are not able to provide routes that are as precise as an on-campus path would require. For example, a user of our application may choose to select ”walking”, ”driving”, and ”bussing” as viable transportation options; the shortest path might then have the user walk to their car, drive to a certain parking lot, then take a bus to the destination. Our Campus Navigator application enables users to obtain routes that are much more detailed and precise than an existing commercial application can provide.

Our implementation of a navigator application calls for much greater complexity than the simplest version of this type of application would. At its core, a simple navigation application would not be difficult to create. The campus would be represented as a Graph structure, with on-campus locations (buildings, parking lots, etc.) stored as vertices of the graph, and transitions between the locations (roads, sidewalks, etc.) stored as edges between the vertices. A simple Dijkstra's search would then be used to calculate the shortest path. However, the wide variety of queries and paths our application must handle requires us to develop a much more complex application.

Our Contribution. Our extension of a simple navigator application facilitates two main areas of research. First, we explore the complexity of handling shortest-path queries with multiple ways of traversing an edge. For example, an edge traversed by walking should have a different weight than the same edge traversed in a car. The correct weight must be chosen when calculating the shortest path. Also, stoplights and traffic may alter the weight of an edge.

In addition to this, we explore the challenge of designing an efficient and intuitive interface for both users and administrators. Our user-friendly interface is robust enough for both inexperienced and experienced students and allows for simple and complex queries built on top of the Google Maps API. The administrator's interface provides an intuitive and simple way to manipulate edges and locations. These design goals are detailed further throughout the paper.

II. Related Work

A. Roadmap

Two previous directions of work have provided inspiration for several of the innovative features of our application. In [1] (Customizing...), a technique is presented to iteratively refine roadmaps at runtime to remove edges that don't meet query-specific criteria. Work in [2] develops the idea of allowing edge weights to be defined by previous edges crossed. We apply this idea in our Route Ordering concept in section IV.

B. Google Maps Applications

The large number of existing applications using the Google Maps API for direction/location display purposes shows that the API is well-suited for such tasks. Here are two applications that utilize the API in a similar way to our application:

Google Maps Route Planner. This application provides functionality for generating custom driving routes based on user input. The user is provided a map, and the ability to select ”landmarks” for the trip. The program then computes a path between each of the landmarks, and displays the combined paths to the user. The generated routes provide distance and time data for each sub-path along the finished route.

Cell Phone Reception and Tower Search. This application overlays cell phone reception data on top of Google Maps satellite imagery, and displays locations of nearby cellular towers. Here, the application retrieves the location of towers from a database of thousands of latitude/longitude points, similar to how our application retrives the locations of campus landmarks from our database to display.

III. Preliminaries

In this section, we will define terms and notations that we use in the paper.

A. Graph

General Description. A graph data structure is composed of a set of vertices *V*, and a set of edges *E* that link pairs of vertices in the graph. In a weighted graph, a value is assigned to each edge (the weight), representing the cost associated with moving from one vertex to another.

Driving Roadmap. One of the most common uses for a graph is a roadmap representation. For example, a roadmap of the state of Texas could be modeled with a graph; each vertex would represent a city (e.g. Austin, College Station, Dallas), each edge would represent a highway connecting the cities, and each edge's weight would be the distance between the two cities. Such instances of graphs are designated ”roadmaps” because of their primary use: finding paths from one place to another.

Dijkstra's Algorithm. Our application's main purpose is to find shortest paths between given start and goal vertices. For this, we use Dijkstra's algorithm, which uses nonnegative edge weights to solve for shortest paths in directed graphs.

Encoded Data. Additional information can be encoded within the graph to let us fine-tune our shortest-path queries. In our application, each edge weight is given information about the current path request, so that different paths can be provided based on the mode of transportation the user chooses.

IV. Design and Implementation

In this section, we describe the implementation of the Campus Navigator, specifically how our method of implementation solves the issues we were faced within this problem. First, an overview of the communication between the separate modules of the application is discussed, along with the internal structure of our application. Then, we detail the steps involved in the construction of our roadmap graph, and show how our application handles the problems that arise due to the complexity of queries we must handle.

A. Communication

The logic within the Campus Navigator application depends heavily on the flow of communication between three separate units: the Web server, the database, and the Query server. The most visible portion from the user's point of view is the Apache Web server, with logic written in php. When the user submits a request (query) to be solved, the php code running on the web server parses the request and inserts it into a table in the database. Upon insertion, the php polls the database periodically, looking for requests. When a request is found, the Query server parses it into a Query object which contains information defining the request (e.g. start and end locations, transportation options). The Query server solves the individual Query by running a Dijkstra's search on the roadmap. This result is then inserted back into the database, where the php finds it, and returns it to the user.

B. Database

The database stores a graph representation of campus as tables of vertice and edges, with mappings that match each edge with a transportation type (e.g. walk, drive, bus) and each vertex with a location type (e.g. building, department, bus stop, parking lot). Multiple mappings can exist for a given vertex or edge. For example, an edge representing a street might map to both ”walk” and ”drive” transportation types, and a vertex representing the Computer Science building on campus would map to both ”building” and ”department”. Upon initialization, the Query server downloads a copy of the database into memory to speed up the roadmap construction.

C. Roadmap Construction

The Roadmap Graph representation of campus is much more complex than a traditional roadmap application, such as Yahoo! Maps. A campus-based roadmap application must provide multiple modes of transportation (e.g. walk, drive, bus) while a traditional roadmap generally provides support for driving routes only. For example, a result path from one side of campus to another might include walking to a bus stop, riding the bus, exiting the bus, then walking to the destination. Or, the path might include walking to a parking lot, driving to another parking lot, then walking the rest of the way. These possible results show that our roadmap must be constructed in a different way than in a traditional roadmap application. Below, we list the techniques that allow our roadmap to handle these complex paths.

1) Layering: In a roadmap with only one transportation type, roadmap construction is simple: all of the vertices and edges in the database translate directly to the vertices and edges in the graph. In contrast, our application has multiple ways of traversing a single edge. If a ”walking” mapping and a ”driving” mapping both exist for an edge in our database, then these two traversal methods should be reflected in the roadmap. In addition, our result paths must be able to switch between transportation types, but only at certain positions (e.g. parking lots, bus stops).

To solve this, we construct the roadmap in layers, where each layer contains edges traversed by one specific mode of transportation.

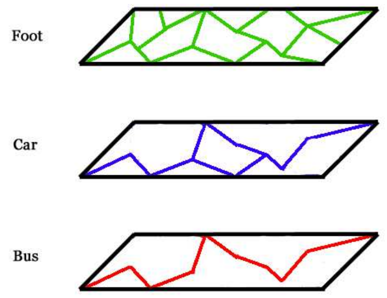
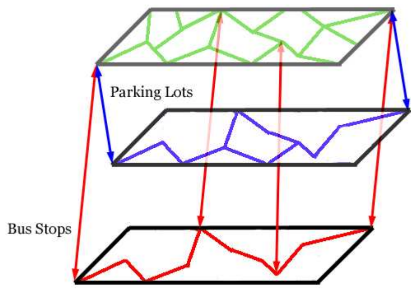


Fig. 1.

To build each layer, we simply iterate through the edge mappings in the database, and add every edge we see that has that layer's mapping associated with it.



e

Fig. 2.

Next, transition edges are added where result paths can cross between layers. For example, the result path can cross between "foot" and "car" layers at Parking Lot transition edges, and between "foot" and "bus" layers at Bus Stop transition edges. This type of roadmap design will constrain the result paths to only change transportation types at designated locations.

2) Bussing Complications: Further work needs to be done to best integrate individual bus routes into the roadmap. Just as the result path should not jump between transportation modes anywhere except at designated areas, it also should not jump between bus routes. In addition, we need to ensure that the result paths always follow bus routes in the correct direction.

Sub-layering. We futher subdivide the Bus layer into sub-layers, where each represents a bus route.

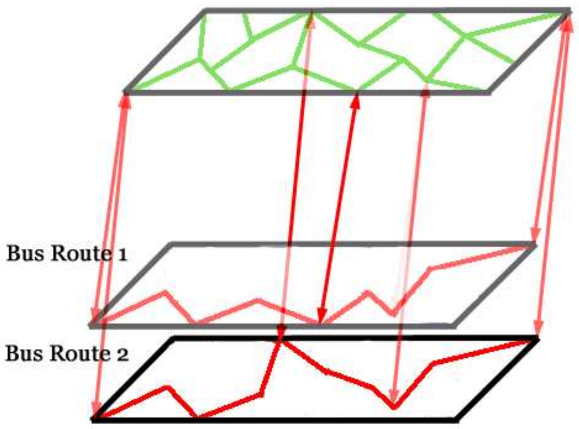


Fig. 3.

Transition edges are added between the Foot layer and the new sub-layer, but only at the Bus Stops that stop along this route. Now, a Bus Stop will likely have multiple transition edges associated with it: one for each route that stops there.

Route Ordering. In routes that include multiple closed loops, the shortest-path search will sometimes want to expand along a bus route backwards, instead of following it in the correct direction. To prevent this, we associate each edge along a bus route with an "order" value, and ensure that the shortest path search must follow bus route edges having sequential "order" values. This requires a slight modification of Dijkstra's algorithm; we give each edge weight in the roadmap knowledge of the previous edge traversed, to perform the sequential order check.

3) Edge Consolidation: A single-layer roadmap is generally composed of high-degree vertices, producing a very interconnected graph. However, we observed that separating our graph into multiple layers results in several unneeded edges.

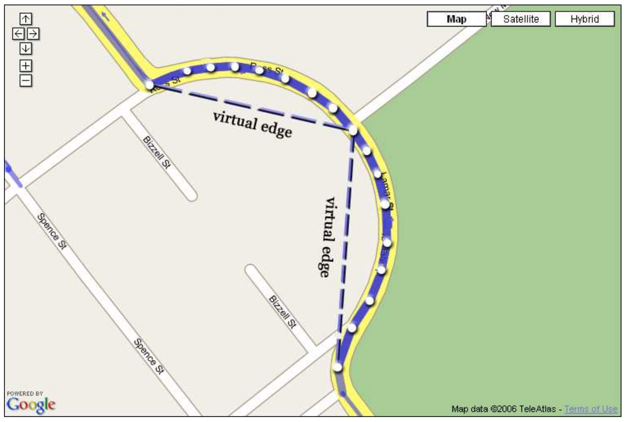


Fig. 4. Several sequential edges can be replaced by a **virtual edge**.

In the above figure, we see that several of these sequential edges in the roadmap can be replaced with a **virtual edge**. To do this, we identify every sequence of two-degree vertices in the graph where none of the vertices have a mapping (building, bus stop, etc.) associated with it. We then replace these edges in the roadmap with a single edge having a weight equal to the combined weight of the existing edges. After the result path is found, any virtual edges are replaced back with their previously existing edges to "smooth out" the displayed result.

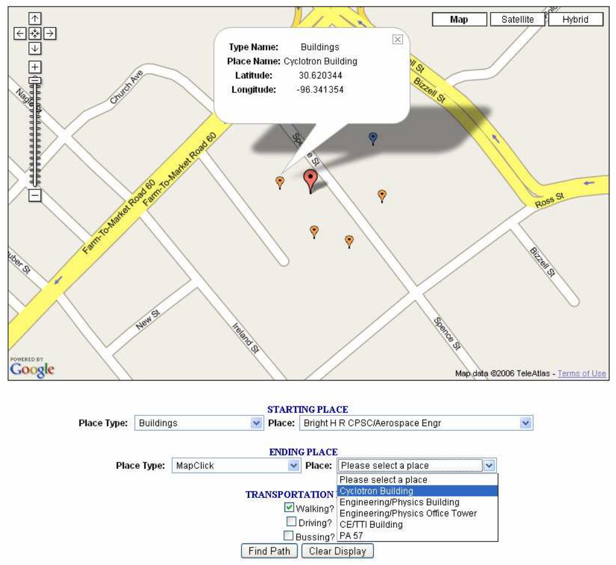
In our campus roadmap, the number of vertices and edges in the graph were each reduced by approximately 30% after performing this operation. This graph reduction allows Dijkstra's algorithm, with running time $O((E + V)logV)$, to execute significantly faster.

V. Interface

In this section, we describe the development of the user and administrator interface. It is crucial that both of these interfaces are intuitive and simple. Using the Google Maps API allowed for visual clarity. First we will discuss the need for the user interface to allow options for both inexperienced and experienced students. Then we will explain display difficulties we encountered and overcame with the administrator capabilities. Lastly, we will discuss the details involved in allowing the adding, removing, editing, and viewing of edges and vertices from the administrator side.

A. User

The user interface is intuitive and simple for both inexperienced and experienced students. It is also adaptable according to their transportation options. The challenging issue here is finding the easiest way for the user to understand the results with regards to the surrounding environment. The user first must choose a starting location. If they already know the type of location they are starting from, then they choose it accordingly. The types of locations available include buildings, departments, parking lots, and bus stops. Then a dropdown list of all A&M locations that are of this location type is displayed. For example, if 'building' is the type of location for which the user is searching, then an alphabetical list of every building on the A&M campus is produced. Allowing the user multiple ways to choose their start and end destination locations, creates a more user-friendly interface.



Type Name: Buildings
Place Name: Cyclotron Building
Latitude: 30.620344
Longitude: -96.341354

STARTING PLACE
Place Type: Buildings Place: Bright H R CPSC/Aerospace Engr

ENDING PLACE
Place Type: MapClick Place: Please select a place

Please select a place
Cyclotron Building
Engineering/Physics Building
Engineering/Physics Office Tower
CE/TTI Building
PA 57

TRANSPORTATION
[✓ Walking?]
Driving?
Bussing?

Find Path | Clear Display

Fig. 5. The map click option allows the user to choose 1 of 5 locations closest to the point on which the user clicked.

If the user is inexperienced and uncertain of their starting location, they can choose 'mapclick' as an alternative way of selecting a start location (see Fig. 5). This allows the user to doubleclick on the map and select one of the five closest locations from which they clicked. In order to provide a clear display, we use the Google Maps markers as locations and associate a different color for each type of location. For example, all of the department locations are purple markers. In addition to this, when the user clicks on a marker, the information about that location is displayed in an info window. In the same way they select their ending destination. Finally they check the methods of transportations that are available to them at the time (Walking, Driving, Bussing). The user can then select the 'find path' button to query for the shortest path or they have the option of clearing the display and starting over.

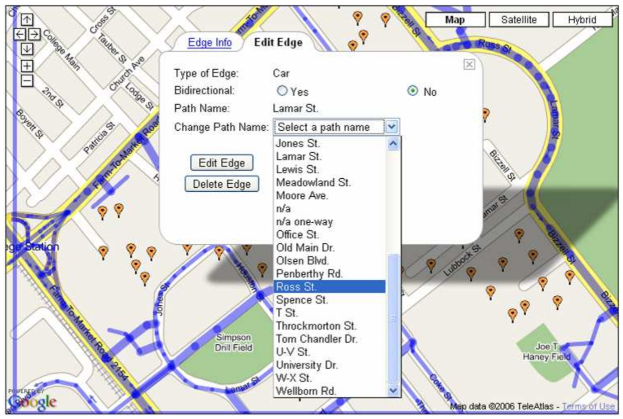
B. Administrator Capabilities

1) Display Difficulties: Due to the size of the A&M campus it is crucial for the administrator to easily manipulate the current locations and transportation paths. Therefore, all vertices and edges should be easily displayed and allow adding, editing, and removing. However our Campus Navigator began to run slower as we entered more campus data. Accessing the database and displaying these vertices and edges generated a Javascript overload that caused the application to run inefficiently. Our Campus Navigator provides a balance between minimizing what is stored in memory (to run faster) while maximizing what the adminstrator sees. Instead of loading all vertices and edges automatically, we provide an intuitive interface which allows the admin to select what types of vertices and edges to be displayed. This allows clarity in viewing the campus, as well as loading the page faster. The administrator does still have the option of clearing the display entirely or displaying all vertices and edges. In addition to only loading and displaying the types of vertices and edges which the administrator selects, we also only load those that are within the bounds of the viewing screen.

2) Edge Properties: To help facilitate a simple and intuitive interface for the administrator we allow a viewing property for both markers and edges. When the admin clicks on a marker the information about that location is displayed in an info window, similar to that of the user's display. In addition to this the admin can double-click on an edge and an info window of the edge's properties can also be viewed.

3) Editing/Removing/Adding: It is important for the administrator to easily manage the system for adding, editing, and removing markers and edges on the map. Our application allows the admin to visit different pages according to the purpose of their task. To facilitate easy input and modification we have both add and edit/remove pages for markers, and similarly for edges. The display properties mentioned above are applied to each of these pages. To add either a marker or an edge, the admin simply double-clicks on the map where it is located and enters the information appropriately.

An admin can easily double-click on an edge and view an info window associated with that edge (see figure below). This allows the admin to easily edit the properties of that edge. There is also the option of simply removing the edge. However, because the edge may be associated with multiple transporation types, the edge is only completely removed from the database if there is no other transportation type associated with it.



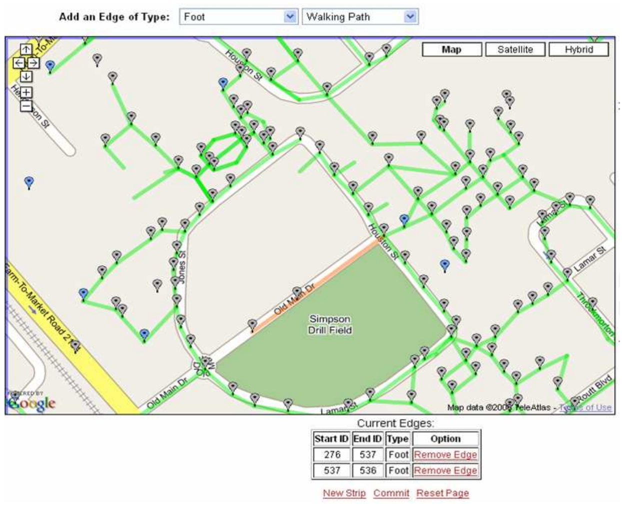
Type of Edge: Car
Bidirectional: Yes [✓ No]
Path Name: Lamar St.
Change Path Name: Select a path name

Jones St.
Lamar St.
Lewis St.
Meadowland St.
Moore Ave.
n/a
n/a one-way
Office St.
Old Main Dr.
Olsen Blvd.
Penberthy Rd.
Ross St.
Spence St.
T St.
Throckmorton St.
Tom Chandler Dr.
U-V St.
University Dr.
W-X St.
Wellborn Rd.

Edit Edge
Delete Edge

Fig. 6. An info window allows the editing and removing of the current edge. If no edge type is associated with the current edge, it is completely removed from the system.

We provide another way to efficiently display markers. The Campus Navigator only shows intermediate markers at the lowest zoom level. This improves the functionality of our application because the intermediate markers are only necessary to be displayed when the admin wants to add edges.


Add an Edge of Type: Foot | Walking Path

Current Edges:

Start ID	End ID	Type	Option
276	537	Foot	Remove Edge
537	536	Foot	Remove Edge

Fig. 7. The intermediate markers are displayed only when the maps is on its highest zoom. This allows the admin to easily add edges.

VI. Navigator Results

The Campus Navigator allows both simple and complex queries using multiple transportation types and provides directions for the shortest path between two locations. Our methodology uses both edge weights and the capability for composite transportation. Both of these help provide a practical approach for inexperienced and experienced students finding their way around campus.

A. Edge Weights

Each edge is associated with a weight which allows for the complexity of handling shortest-path searches with multiple ways of traversing an edge. Weights are dependent upon the time it takes to traverse the edge. For instance, a driving path will have a smaller weight than a walking path (See Fig. 8).

B. Composite Transportation

The Campus Navigator allows for the changing of transportation types while in route, according to the user's preferences. There is a weight associated with transitions between two edges with different types of transportations. The time spent at a bus stop waiting for the bus to arrive needs to factor into the weight of selecting that edge over another, such as walking (See Fig. 9).

VII. Conclusions

Future work on this project will focus in three main directions: new navigation features, time-specific queries, and interface improvements. Work in the area of navigation improvements will include both the integration of on-campus building layouts and paths to off-campus locations (e.g. eating establishments, hotels). These improvements will help target a wider audience. The second area of research will allow the adjustment of edge weights based on the time-period a user selects. For example, available parking locations change during large on-campus events, such as football games and graduation. Also, additional parking lots are accessible to most parking permits on nights and weekends. Work in this area will provide more precise results when querying for the shortest path, and give the user more control over their query. Lastly, interface improvements, such as a custom set of images for the campus, will provide greater clarity for the result path.

The Campus Navigator provides on-campus directional queries that are more specialized than those in commercial applications, such as Google Maps and Yahoo! Maps. The application provides an intuitive interface that allows the user to perform both simple and complex queries on the Texas A&M campus, covering a variety of methods of transportation. In addition, the design of the administrator interface facilitates simple, yet detailed, manipulation of edges and locations.

VIII. Bibliography

Simulating Protein Motions with Rigidity Analysis, S. Thomas, X. Tang, L. Tapia, and N. M. Amato, Technical Report TR05-008, Parasol Laboratory, Dept. of Computer Science, Texas A&M University, Sep 2005.

A Path Planning-Based Study of Protein Folding Pathways with a Case Study of Hairpin Formation in Protein G and L, G. Song, S. Thomas, K. A. Dill, J. M. Scholtz, and N. M. Amato, Proc. of the 7th Pacific Symp. on Biocomputing (PSB), pp. 240-251, Jan 2003.

Using Motion Planning to Map Protein Folding Landscapes and Analyze Folding Kinetics of Known Native Structures , N. M. Amato, K. A. Dill, and G. Song, J. of Computational Biology (JCB), 10(2):239-255, Nov 2002. Also, in Proc. of the 6th Int. Conf. on Computational Molecular Biology (RECOMB), pp.2-11, Apr 2002.

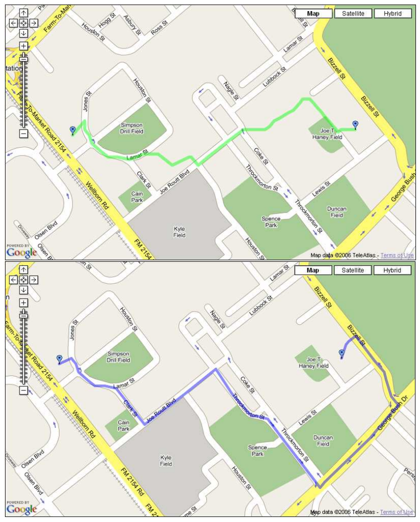
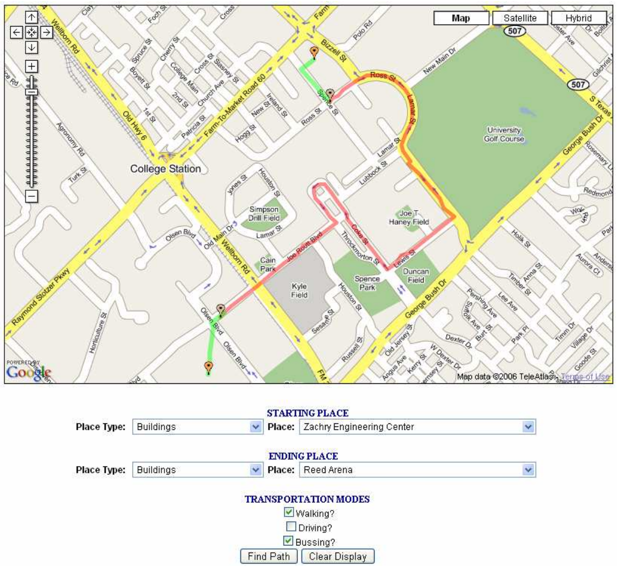


Fig. 8. An edge traversed by walking has a different weight than the same edge traversed in a car.



STARTING PLACE

Place Type: Buildings Place: Zachry Engineering Center

ENDING PLACE

Place Type: Buildings Place: Reed Arena

TRANSPORTATION MODES

✓ Walking?

Driving?

✓ Bussing?

Fig. 9. In this case, taking a bus and then walking produces a more efficient path than walking the entire way.