EG 11112
Introduction to Engineering System II
Learning Center Section 2
Jason M. Grant

Campus Map

Group 1:
Tallis Bowers, Brittany Harrington, AJ Pucci, Pat Myron, Katelyn Miller

30 April 2014

Abstract

The motivation behind this project is to create a program that is applicable to the typical student and that can be utilized realistically in everyday life on campus. The campus map project is a good choice because it could be of use to every student living on Notre Dame's campus. The main idea for this campus map project is to create a program and generate a GUI that displays a virtual map of Notre Dame's campus, calculates the shortest path around campus, and displays both the path and its distance and time to a user. Once the user inputs their starting and ending buildings, the program will calculate the shortest path between the two and trace this path onto the map, showing the total distance and time, based on chosen mode of transportation, as well. The approach to this project is to create a program that utilizes Dijkstra's algorithm to calculate the shortest path between two buildings. The program is based upon a system of nodes that represent all intersections of pathways being utilized on the map. By comparing the values of distances for each path leading away from a node, the algorithm is able to find the path that corresponds to the total shortest distance between two given nodes.

Once the program was completed, the group tested all pathways and compared the distances computed through the program against distances found on Google Maps. Overall, the program had a percent error of 6.97%. This value is extremely reasonable due to assumptions made that all paths are straight, all nodes for intersections are in the right place, and accounting for rounded distance values. To show that this algorithm was the most effective at choosing the shortest path, the group added an additional section of the program that accounted for traffic. Weights were added to certain nodes in congested areas (i.e. outside DeBartolo Hall), so this new program would choose an alternative path that travelled a longer distance, but with a more accurate representation of time. Each time the group ran the program for a path with and without traffic, the option without traffic always found the shorter distance. Because this program calculated the shortest path between two buildings on every run, it fulfills the design statement and can be considered a success.

Overall concept

The design statement for Group 1's campus map project is to create a virtual map of Notre Dame's campus that calculates the shortest path around campus and displays the path and its distance to a user. The user is able to pick a starting building and ending building, and based on his choice of building, he is also able to choose the doors from which he wants to exit and enter. Based off these inputs, the program finds the shortest path between them and traces this path on the campus map. It also displays the distance associated with this shortest path, in feet, to the user and the time it takes to travel, based on his mode of transportation. The program also has an option to account for traffic, which gives the user an alternative route avoiding congested areas and allows for a more accurate representation of travel time. The program also allows the user to cut through buildings to ensure the quickest possible path.

Figure 1 is a flowchart that explains the run of the main program.

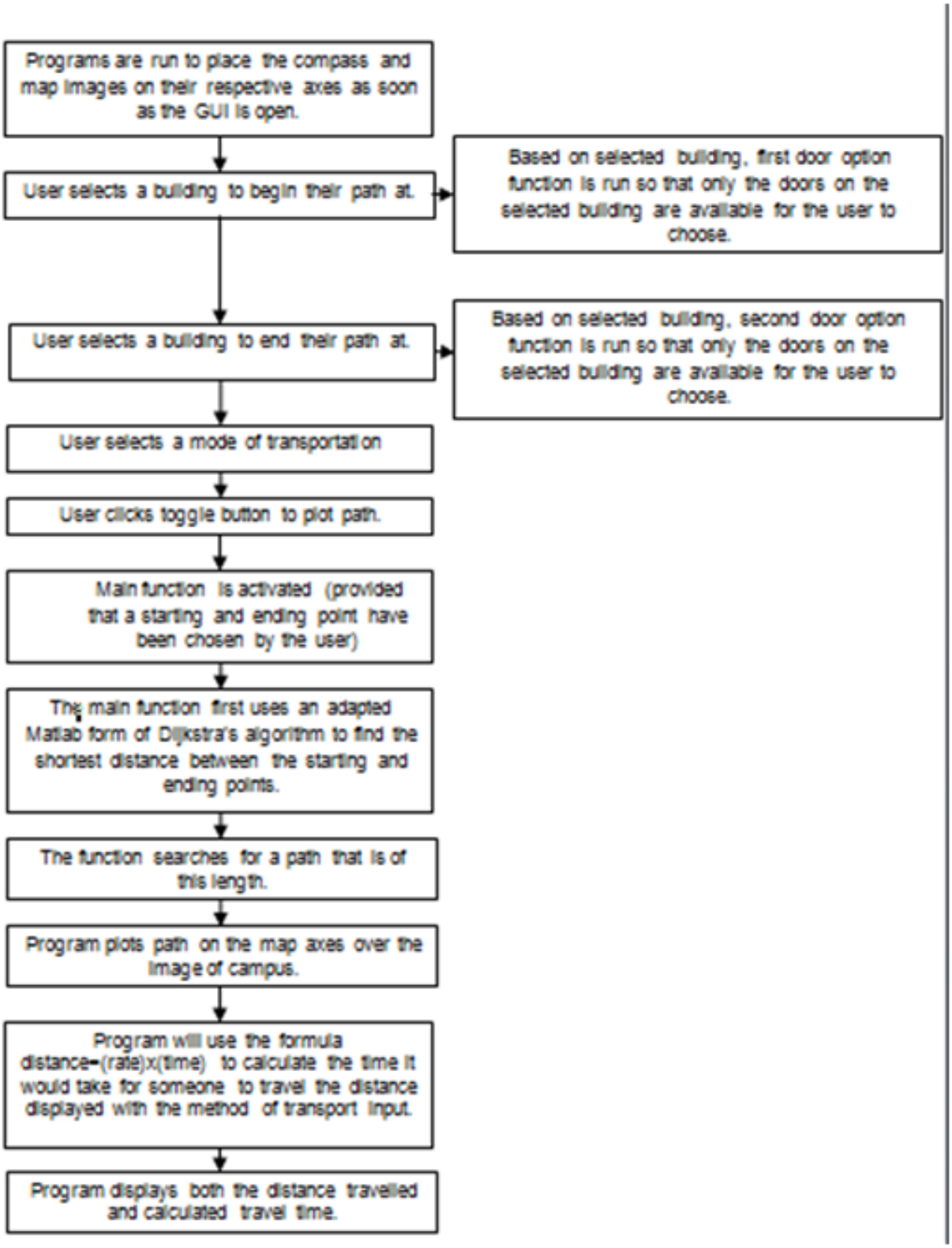

Figure 1

One design change that occurred to get to the final design was the addition of an option to account for traffic. Originally, the program only calculated the shortest path based on distance

and though it calculated the time travelled, it was not always an accurate representation of time because the shortest path often leads through congested areas. The group added weights to nodes in the matrix that corresponded to popular paths on campus (i.e. outside DeBartolo and La Fortune) so that when the algorithm searched for the shortest path, it chose a new path that had the lowest total weight, bypassing the congested areas. Before the group added this option, there was also no comparison to show that the program really was the most effective at choosing the shortest path between two destinations. Once an option for traffic was added, it was easy to see that the algorithm with traffic always chose the shortest path (distance), since the total distance covered by the path with traffic was always longer than without.

Underlying Concept

The underlying concept behind the whole project was to understand how mapping programs use a mathematical algorithm to suggest a path between two points for a user. In doing this, the group first had to learn the basics of a so called "shortest distance" algorithm. Though there were many options for this, the group decided to use Dijkstra's algorithm because after learning a little bit about the way the algorithm works, it was decided that it would be fairly easy to adapt into Matlab code. The group found for the most part that this was true of the logic behind the algorithm. As do many shortest distance algorithms, Dijkstra's algorithm simplifies the area being mapped into an interconnected web of points and lines. In its simplest form, the program represents such a web with over 140 points and a few hundred lines connecting those points. According to the algorithm, once presented with a starting point and an ending point one should examine the points connected to the starting point with a line. The value of the distance between the first point and each of these secondary points is than assigned the secondary points themselves. After each secondary point has been assigned a value, one moves to the secondary point with the smallest assigned value and repeats the process for the group of points connected to this point assigning values of the distance of this secondary point from the starting point plus the distance of the each of the points in the third group from the secondary point. If this second step were to assign a value to a point that is already assigned a smaller value, the new value would be forgotten for the older, lesser value. This process is continued by moving to the point with next smallest value in each step. The algorithm ends when the ending point has been assigned a value.

The program uses a 1x141 vector to store the assigned value of each point and a 141x141 matrix to store the numerical representations of the distances or lines connecting each point. If two points n and m are not connected by paths on the actual map, then a value of infinity is placed at the locations (m,n) and (n,m) in the large matrix. A value of infinity is also placed at every point in the 1x141 vector until a smaller value can be assigned to each position while running the program. The program deviates slightly from Dijkstra's algorithm in that it does not stop when the ending point has been assigned a value but rather it continues the process until every one of the 141 points has been used to assign values in the vector. The group found this to be more effective as stopping once the ending point was assigned a value did not always guarantee that that value was the smallest possible value. This did in theory prolong the length of time needed to run the program, but it was a positive choice for our project

as the run time now is still very short and we can be much more confident in the program choosing what is actually the shortest path between two points.

It is clear that this algorithm only determines the numerical value of the distance between two points, in order to plot the correct path in a GUI over a map of the portion of campus that was used for this project, the group used their own thoughts and knowledge of Matlab. The group decided to put the map of campus in an area on an axis that is equivalent to a 100x100 square, so x and y coordinates could be found as percentages of the full picture. Then the group used measuring abilities to create two vectors that stored the x and y coordinates of the locations of each point being used on the map. The program then connected only those points whose total path length was equal to the shortest distance determined with Dijkstra's algorithm. This created a few logistical problems. The group had a hard time making sure each of the points had exactly the right coordinates for it to show up at the correct spot on the map, and the program was unable to graph the slight curvature of some paths as all of the created paths were a series of line segments.

The only equation used throughout the process was used to determine the time it would take a person to travel the distance generated by the algorithm as well. In the end, the group was extremely happy with the finished project.

The equation to determine distance travelled is defined as:

$$\text{Distance} = \text{rate} \times \text{time} \qquad (1)$$

Where distance is measured in feet, rate refers to the rate of travel, measured in feet per minute, and time is measured in minutes.

In creating the program, the group had to make a few assumptions. The greatest of which was that Google Maps, used to measure the distances of paths, was accurate to within a few feet for each measurement. The group also made the assumption, when determining the time needed for a trip, that a person would use the same form of transportation throughout their entire trip despite graphing some paths through buildings. It is of course not feasible for a person to actually ride a bike or long board through a building. Finally, for purposes of comparison, the group created an option in our program for users to account for traffic when finding a path. In this algorithm it was assumed that the weight added to certain paths would be a good approximation for the added time one would need to walk those paths during heavy traffic. This was a big assumption because it was almost entirely opinion based. In comparing the two forms of the program, the group determined the original shortest path form to be the most useful form of the program. While it may only consider distance needed to travel a path and not the time it will take one to realistically walk that path, the group does not believe traffic is much of a hindrance to regular commuting at most times of the day.

GUI Tool

Figure 2 shows a screenshot of the final GUI, with the shortest path between Nieuwland, upper West door, and Stinson-Remick, lower East door, with the option to account for traffic selected.

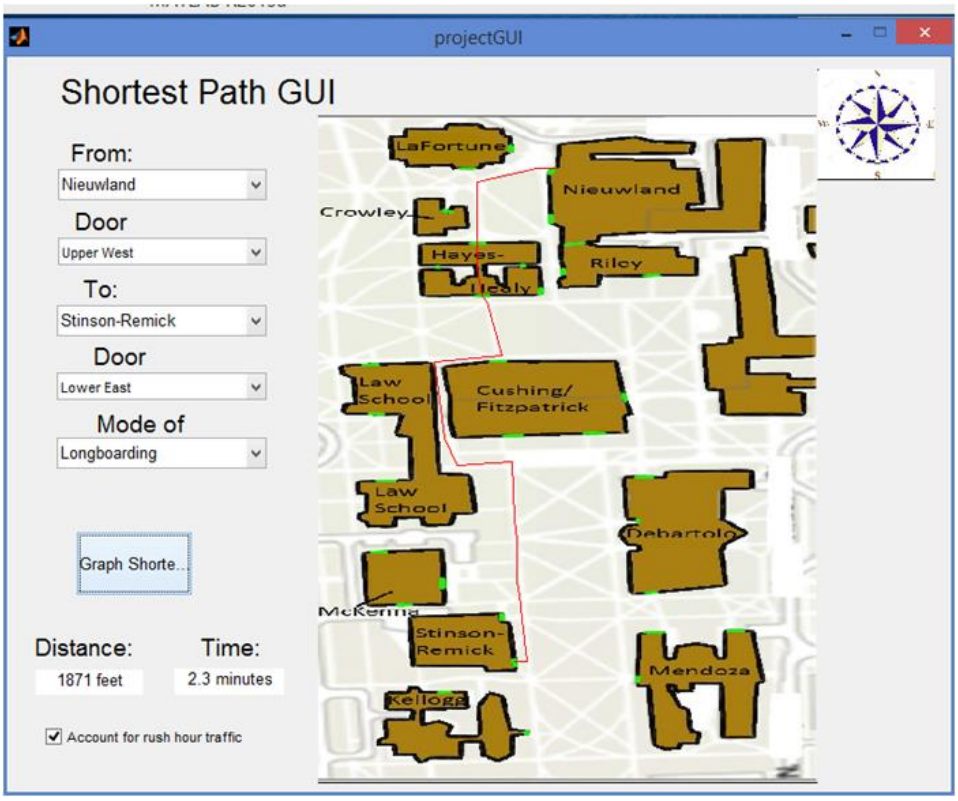

Figure 2

The GUI tool is set up this way because it requires a lot of user inputted options. Since the user was allowed so many options, popup menus were used to create all of the options to the user from the doors of each building to the mode of transportation. It was all dependent on the user and was able to create the best generated plot from the popup menu choices. The checkbox is the only other input on the GUI and it is used to create a weight matrix for the plot in order to give a more alternate route in case of traffic. The checkbox is another user inputted option. Since it has only two options, it is a checkbox instead of a popup menu.

The group created a few different subfunctions to better organize the code needed for this program to run properly. Subfunctions were created to hold the large 141x141 matrix and the matrices for the x and y coordinates of the points on the graph. The group also created subfunctions that allowed the door drop boxes to change the options they presented user with depending on their choice of building. Finally, the group used functions distinct from the main function run in the GUI to put the image of the portion of campus and the compass into the GUI as soon as it is opened.

Discussion of performance

A major prediction made by the group is that the algorithm should always choose the shortest path between two points. The group's predicted error, comparing the distances generated by the program to distances found on Google Maps, was around ten percent. This seemed like a reasonable margin of error, taking into account the size of intersection points, the estimation of door placements, rounded initial distance values from Google Maps and the assumption that all paths are straight. Another prediction made by the group is that the traffic option always provides a quicker path that is not the shortest distance, but the fastest time.

Table 1 shows the comparison of the distances generated through the program to distances found on Google Maps for ten possible paths.

Table 1

From → To	Distance from Google Earth (ft)	Distance by Map Algorithm (ft)
McKenna (N) → Crowley(N)	1617	1388
Mendoza(NW) → Hayes-Healy (W)	1450	1412
LaFun(S) → Stinson-Remick (UE)	1667	1693
Nieuwland (UW) → Stinson-Remick(UE)	1755	1741
LaFun(E) → Debartolo (N)	1017	1214
Hayes-Healy(N) → LaFun(S)	246	246
Law (M) → Mendoza(NW)	1023	977
Debartolo(W) → Kellogg(N)	827	897
Riley(S) → Fitzpatrick(E)	394	426
Crowley(N) → LaFun(E)	289	318

The data in Table 1 generates an error of 6.97 percent (comparing the experimental data to the actual), which is well below the group's predicted error of ten percent. The paths with distances that varied the most from Google Maps also cut through buildings such as DeBartolo and Hayes-Healy, which is an option that Google Maps does not provide. Cutting through the buildings makes for a shorter total distance and could account for some variation seen in the test results.

To ensure that this algorithm is the most effective at choosing the shortest path between two buildings, the group created a sub-program that allows the user to account for traffic. The

prediction was that this program would provide a more realistic representation of time, but where distance is concerned, it would always choose a longer path than the initial algorithm.

Table 2 shows the comparison of the distances generated through the program without traffic, to the distances generated through the program with traffic, to the the distances found on Google Maps for ten possible paths.

Table 2

From → To	Distance from Google Earth (ft)	Distance by Map Algorithm (ft)	Weighted Algorithm (ft)
McKenna (N) → Crowley(N)	1617	1388	1617
Mendoza(NW) → Hayes-Healy (W)	1450	1412	1508
LaFun(S) → Stinson-Remick (UE)	1667	1693	1693
Nieuwland (UW) → Stinson-Remick(UE)	1755	1741	1753
LaFun(E) → Debartolo (N)	1017	1214	1510
Hayes-Healy(N) → LaFun(S)	246	246	246
Law (M) → Mendoza(NW)	1023	977	977
Debartolo(W) → Kellogg(N)	827	897	897
Riley(S) → Fitzpatrick(E)	394	426	468
Crowley(N) → LaFun(E)	289	318	318

The data in Table 2 supports the claim that the shortest distance algorithm always produces a shorter path that the weighted algorithm which accounts for traffic. On all of the ten paths, the distance for the initial algorithm is shorter than when traffic is added.

Some assumptions could be adjusted to lower the overall error of the program. Right now, the node for each intersection was chosen carefully, but they were not measured to be the exact center of the intersection. By making each of these nodes more exact, the distances

between nodes would become more accurate. Also, the group could add more nodes along curved paths to account for the increased distance that curvature provides. Each path in the current program is assumed to be straight, but looking at the map, this assumption is not always true.

One change that could be implemented to improve the project would be to make the program adjust for forcing people to walk through buildings. As it is set up now, the mode of transportation does not change when people cut through buildings. This is unrealistic for every mode of transportation besides walking and running. It would provide a more realistic representation of time if the program forcefully changed the mode of transportation to walking each time the path cut through a building, or if it only allowed to cut through buildings if they selected the walking mode of transportation. More nodes could also be added along curved paths to account for the curvature. Right now, all of the nodes exist at intersections between paths, and the paths are all recorded as straight lines, which is not completely accurate. Some of the paths are indeed straight, but there are also some that have a curve to them, which makes the distances slightly inaccurate.

Though there are improvements that could be made to make the program more accurate, user-friendly, and run smoother, the data collected proves that the algorithm always calculates and displays the shortest path between two buildings on campus. The program fulfills the initial design statement and can ultimately be considered a success.

References:

Compass Rose. n.d. *Dreamstime.com*. Web. 10 Apr. 2014. <http://www.dreamstime.com/royalty-free-stock-photography-compass-image289767>.
Map.nd.edu
Maps.google.com
Yan, Melissa. *Dijkstra's Algorithm.* n.p. n.d. Web. 20 Feb. 2014. <http://math.mit.edu/~rothvoss/18.304.3PM/Presentations/1-Melissa.pdf>.