(Extended Abstract)

Routing people and vehicles in enclosed spaces

Alexandre Santos Henriques

Supervisor: Prof. Renato Jorge Caleira Nunes

October 2014

Abstract

The subject of this thesis is the routing of persons and vehicles in closed spaces. The objective is the orientation from an origin point to a destination point, while aware of the layout of the building and a series of restrictions that can be defined by the users.

Despite the fact that most current smartphones combine GPS with different Wi-Fi positioning systems, the number of applications that use these technologies with the goal of routing their users in closed spaces is still very small and with very limited features. Currently there is no knowledge that any of these applications work in Portugal, being mainly restricted to a single country, mostly the USA, or to a specific type of building or even a single building.

A study of the algorithms for the routing of people and existing location systems will be conducted, although the main focus of this thesis will be on the routing component. The goal is to develop an open and flexible solution that could be easily adapted to different spaces such as parking lots, malls, hospitals and other public spaces. An application for the Android operating system was developed, TravelBuddy, which could be used in multiple spaces to achieve this goal.

TravelBuddy presents itself as the universal solution to be adopted by everyone, since by using low resources of any Android device, it solves the presented problem while quickly giving its users easy and clear instructions of routing in different languages, despite the country in which the users reside or the geographical location of the buildings.

Key-Words: Routing, Application, Android, Building

1. Introduction

It has been a trend in the recent years to build more and larger buildings, which makes difficult the task of the common person to guide himself in these spaces. While there has been a great evolution in technology in the recent years, incredibly it isn't known any open application that helps solve the problem of orientation in enclosed spaces, and that is precisely what this thesis seeks to address.

With that being said, the goal of this thesis is to create an open and generic solution, so that it can be used in any space (thereby not limited to parking lots and malls) but also be applied to business buildings, hospitals, or any other building regardless of its size. It's also part of the goal that the user of the developed solution can set restrictions to his course. For example, in the case of a person with limited motor capabilities there should only be considered pathways where there's not required to climb stairs, being the application responsibility to calculate a path for the user that meet this requirement.

To provide the desired flexibility, it was created a specification for buildings in XML that will represent all the structure and features of it, such as its dimensions, inputs, outputs, and other relevant information in the most rigorous way possible. This way it's possible to convert the layout of any

building in a file that can be read and used by the developed android application, TravelBuddy. So when a user enters a new space, it'll be contacted a server associated with the infrastructure that sends the XML specification to the application and consequently this is the information that will be used to assist the user. Alternatively, it can be used a global server for storing the specification of multiple spaces, and the user only needs to download the desired maps.

The emphasis of the thesis is the description of space, the user interaction and routing. It's not an objective of this thesis to detail the detection position mechanism (in the context of the thesis it can be assumed that there are mechanisms that provide this location information to the application, i.e. the application does not have to worry about that aspect). Anyway a survey of possible solutions will be done in this area which eventually may apply directly, but the main emphasis is not in this area.

2. Related Work

The study of related work revealed that the indoor positioning systems have been, in the past few months, widely supported by some of the major software companies in the world. This shows that not only this area has a great potential but also that soon we can expect more available resources that can be used for this purpose, leading to an increase of better applications of this kind. Still, some of the most promising systems are still under development, and most of them won't be available to the Portuguese market any time soon, which leaves us with a lack of solutions of this nature in Portugal. Even in the U.S.A, the most likely country nowadays to have more and better solutions of this type, none of the referred applications came close to completely solve the problems exposed in this dissertation, thus being necessary the development of a solution that succeeds where most of the others still fail. It's important to refer that during the search for solutions to the exposed problem, not even one solution was found that combined simultaneously the routing of people and vehicles in closed spaces. With that being said, we can conclude that it will be necessary to create a proper solution for the given problems from scratch.

Since there's a need to create a new solution, it's essential to explore the related work about routing algorithms. Over the last few years remarkable scientists have studied, developed and refined this kind of algorithms. Between the most noticeable solutions we can find the A* [1] [2], Bellman-Ford [3] [4], Floy-Warshall [5] [6], Dijkstra [7] and Johson [8], which are the ones that have been studied in detail in the context of this dissertation. But it turns out that all these algorithms were developed for their own applications, and therefore have their own limitations in the scope of what is required for this thesis; anyway it was still important to study them because they all introduced some very interesting ideas and concepts that can be adapted to solve some of the problems at hand.

3. System Architecture

The first time a user runs the application in a new space where it hasn't been before, it'll communicate with the space's dedicated server (as seen in figure 1) to download new maps or update them.

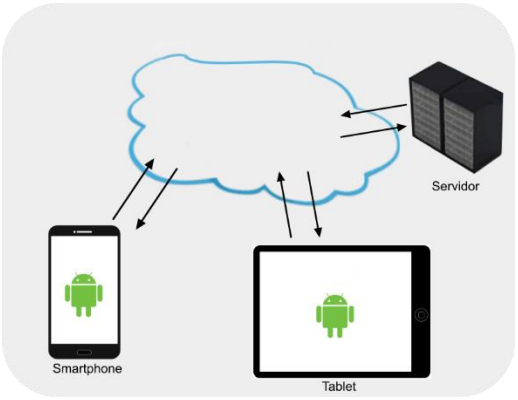


Figure 1. Interaction between the application and the space

Every time a new XML file is received, the application has to parse it in order to extract the information about that building. Once the information is extracted, it'll be stored in appropriate Android data structures, such as treemaps and arraylists, in order to be possible the access further ahead. That information will be used not only to the routing algorithms, but to define in a custom way the majority part of the application menus too, since every single building/type of building will have its own different categories, divisions, and so on.

The first action of the user when the application starts, it's to choose if he wants to receive orientation while walking or while driving a vehicle. From that moment on, all the menus shown to him are specialized in one thing or another, being possible to the user switch between them anytime.

Known the mode of the application and the user's current location, as well his desired destination, the application will run its algorithm (depending on the chosen mode), that by using the previously stored information about the building, allows to solve that routing problem. The solution will be presented step-by-step in the form of text in the application screen to the user.

Regardless of the orientation mode, the steps presented to the user are always of the same type with more or less detail (as we'll see further ahead), since the important part of any orientation given is to be clear about when someone should turn where along his path. So, in TravelBuddy, the user will receive new orientations every time he has to decide between multiple paths, despite having or not to change his current direction.

3.1. Walking mode

While in the walking mode, the user can choose his destination by category or by name (the first option works as a filter that displays only the destinations by name of the category chosen), however the application will not allow the user to choose his destination if he hasn't defined his current location first.

In order to a user define his location, he'll have to press the "my location" button (the crosshair button) present in any menu on the walking mode. The user will have then two different ways to define its own location that he can choose by selecting the corresponding tab. In the first method the user has to type

its current location name or id, on the other hand, the second method consists in aiming the android device's camera to a QR code that will automatically validate the user location in the application logic.

All set, the algorithm runs and the orientations are given to the user. So it's time to understand the algorithm.

The routing problem of people in enclosed spaces is divided into two parts, the first of which represents the orientation of a person on one floor of a building, and the second, the same orientation between different floors. Thus there is a graph to represent each existing floor of the building and an extra one to represent the building's height, thereby connecting all the other graphs.

For the representation of a floor of a building, it's only taken in account to the graph the existence of corridors and intersections, being a corridor all the common area outside of the divisions, and an intersection any place where two or more corridors intersect. So it's understandable that corridors work as edges of the graph and intersections as nodes. The following figures shows it in a simpler way. The first represents one floor of a building from a user's perspective, the second one represents the same but in the point of view of the algorithm, and the last one represents again the perspective of the algorithm, but this time finally as the form of a graph.

Figure 2. A building's floor from a user perspective

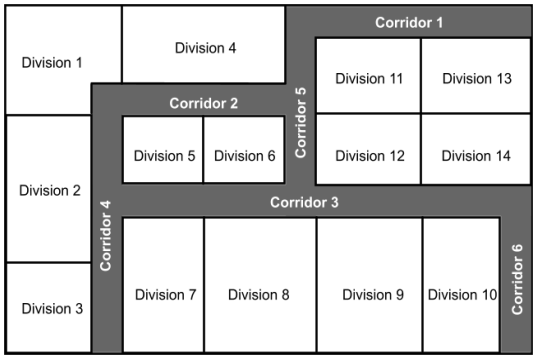
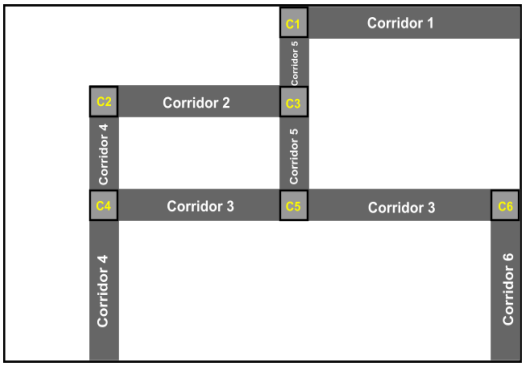


Figure 3. A building's floor from the application perspective



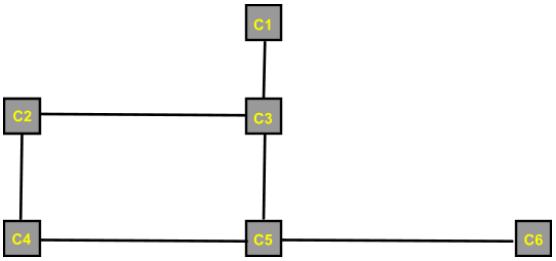


Figure 4.A building's floor in the graph form

As for the representation of the connections between the various floors of the building, the graph only considers the existence of access between floors, which can be elevators, stairs, ramps, etc. As such the nodes of the graph are represented so many times as the number of floors that they serve. The edges, on the other hand, connect a node to all the other nodes corresponding to different floors served by the same access and to all the other nodes on the same floor. There are two figures below that represent an example of what was explained.

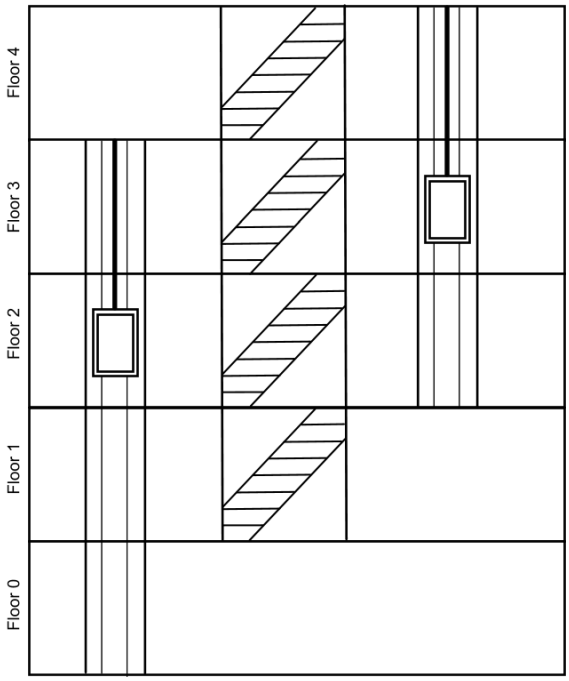


Figure 5. Vertical cut of a building

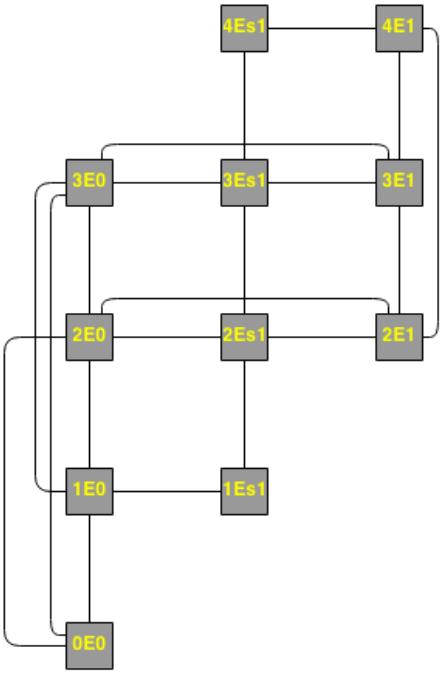


Figure 6. Height graph of a building

To proceed with the explanation of the algorithm implemented, it's first necessary to distinguish the different possible cases in which the same can be applied. It is important to note that although both graphs above never consider the existence of divisions, it's between them that the users want to move most of the time.

With that being said, there are situations where the user's origin and destination:

A. Are the same;
B. Are in the same corridor on the same floor;
C. Are in different corridors on the same floor;
D. Are in different corridors and on different floors;

The algorithm starts by checking if the source and destination are the same. If they are the same, problem A is solved. Otherwise it will then be verified if the origin and destination's floors are the same. If they are, the problem is either type B or C, otherwise the problem is type D.

In case that the floor of origin and destination are the same, it will be made a new check to verify if they are on the same corridor, and one can thus distinguish the situations B and C. In situation B, the problem is trivial, because if the person is in the same corridor that his destination, it's only needed to calculate the distance to travel from where the person is to the access of the destination (door).

In situation C, it will be necessary to run Dijkstra's algorithm to the graph of the floor in question. By following each step belonging to the solution obtained, the person will be moving from intersection to intersection until reaching an intersection that connects to the destination corridor. From that moment, the problem is then type B, thus making it easier to resolve.

Finally, if the problem is type D, the application will then run the Dijkstra's algorithm on the graph that represents various floors, as many times as the number of combinations of accesses between floors present in the origin floor and the destination floor, being the chosen solution the one with the shortest path. From that moment on, the person has as the intermediate destination the first step of the path obtained by running Dijkstra, and therefore, the problem is now of type B or C, meaning that the respective solution will be executed. When the waypoint is reached, it will be replaced by the next step of the result of Dijkstra and the process is repeated until eventually it reaches the last step of Dijkstra for floors. At that moment, the destination will no longer be replaced by the original, since the person has to be already in the same floor as his destination, so it will be solved by the last time a problem of type B or C.

After running the algorithm, it's time to present to the user the indications of how to proceed to the next point in his path to the chosen destination. Messages after messages are displayed every time the user presses the "proceed" button. While that's done, the user is confirming that he had already reached the last destination given by the application, so that's his current location and it's now ready for a new indication. That being said, if a TravelBuddy user runs it all the time that he is in a certain place, it'll only need to inform the application about his location one time, since all the others will be determined automatically as the user travels from one destination to the next one.

It's also important to understand that the indications given by TravelBuddy are the most complete possible, in a way that it's impossible to misunderstood them, but it's also impossible to be confused with the amount of information. For example, a real orientation given path by path by TravelBuddy can be:

"Walk in the current corridor for 20 meters, in the XPTO's direction, until you reach the crossroad (C3, with the lion statue)" / "Turn left and walk for 20 meters to the elevator." / "Go up in the elevator to the 2nd floor." / "Turn right and walk for 10 meters, in the QWERTY's direction, until you reach your destination." / "You arrived your destination."

3.2. Driving mode

From the moment the user is in the driving mode, he will be promptly asked by its destination category. Unlike the previous mode, there is no other alternative, and the category selection no longer works as a filter, but as the final destination of the user, since every category's associated with a parking lot section.

Another big difference from this mode to the last one is that the user will no longer inform the application from its location, since it's not the best idea to interact with any kind of device instead of being focused on the driving itself. Instead, the information about the user location will be collected automatically by the application, being necessary for that, that the device's Wi-Fi's turned on. It's assumed that it will be an identity (or more) that will continuously broadcast to the application the information about the nearest crossroad, and how many empty parking slots are in that area.

That being said, from the moment the user chooses the destination category, the application runs the respective algorithm to calculate the path between the user current location and the desired destination, and presents to him the first step of the calculated path. As soon as TravelBuddy receives a new message (a message where the current location is now different from the one received in the previous message), a new step is displayed to the user.

Once in a parking lot, place in which the vehicles routing algorithm will run most of the time, all floors are accessed sequentially by ramps, it was decided that it would not take more than a graph for the entire representation of a such space, unlike what was seen in the previous case where there was a graph by floor and an extra one to represent the link between the multiple floors.

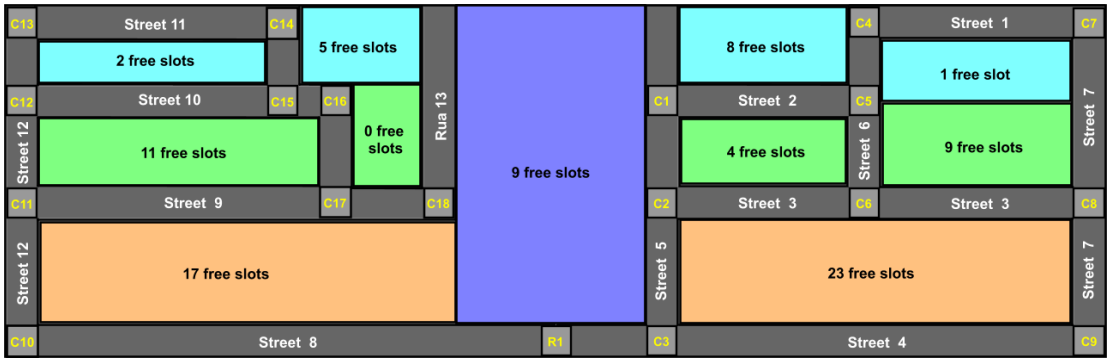


Figure 7. Parking lot layout of a building

Thus, the graph has the streets presented on the parking lot as edges, and the junctions between them and ramps like nodes. The weight of all edges is always equal to one, except where at least one of the nodes of the edge is a ramp, wherein the weight will be equal to two. This happens this way because, although it isn't the goal to find the fastest way to the destination, it is to find the path that takes the user to perform fewer turns, and driving in ramps is a synonym to perform more complicated maneuvers that may take longer, reason which the respective weight must be compounded in the graph.

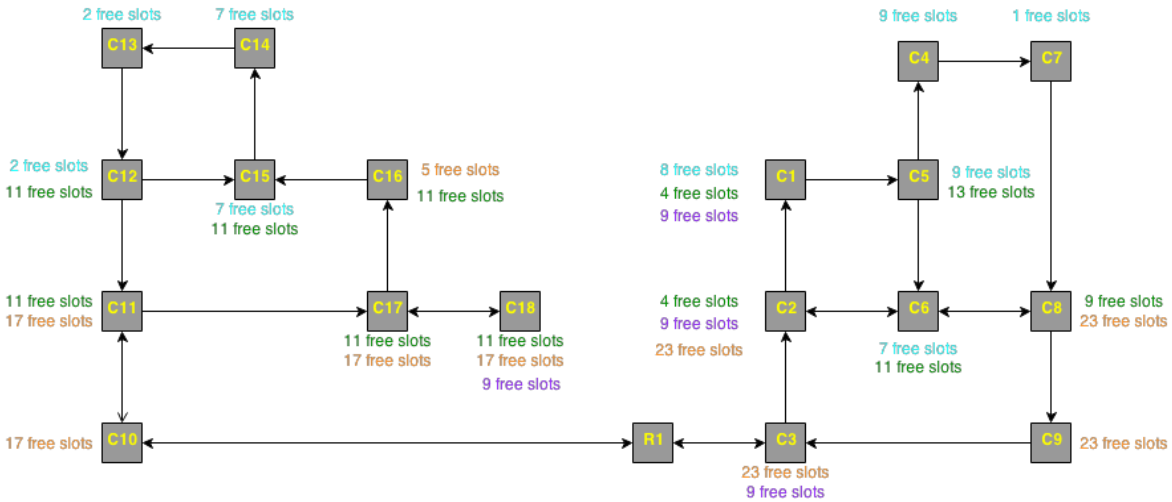


Figure 8. Graph of a building's parking lot

It is also important to remember that being this a place subject to the application of the Highway Code, the graph created will be directed to comply with the directions of the streets. It's also a must to retain that the user's destination will always be a parking area constituted by several crosses, although a cross can simultaneously belong to different zones. Obviously it isn't enough to reach an intersection belonging to the destination in order to the algorithm find a solution, since it is a parking lot, it's also necessary that there are free slots in order to the algorithm terminate.

The algorithm always starts by calculating the Dijkstra between the source and the destination, where the destination is considered the cross more far away belonging to the possible destinations. Each execution of the algorithm checks if the user is already on the destination and if there are empty parking slots available.

In the event that the place where the user is, isn't part of the target area, it will be checked if at least it belongs to the path calculated by Dijkstra's algorithm. If not, the user is clearly outside of the calculated route, so this is recalculated being the current position of the user the new origin point. Otherwise it simply proceeds to the next step of the solution.

If the place where the user are located belongs to the target area and has free parking slots the algorithm comes to an end, but in the case where there are no free slots the algorithm continues in order to the user moves to a next possible destination location which may have free places.

In the worst case scenario, all the possible target locations has no free slots available, so the algorithm terminates and the user is forced to choose a new destination or simply wait for a free slot in the desired parking.

After the user park his car and give that information to the application, it'll be informed that the car location has been saved, and will be presented with the opportunity to add some optional notes about the car location, in order to find his car easily in the future. Once that's done, the user will be directed to the nearest elevator or stairs to reach the main building.

3.3. Other Functionalities

TravelBuddy counts with a fair amount of bonus features as we're about to see.

The first and most noticed one will be the Favorites menu, since its button is present in every menu of the walking mode. In this menu the user can store his favorite places in order to make it easier to find them next time he needs it. All the new custom locations, only available by scanning QR Codes, can also be accessed in the favorites menu after the user finds them.

Another fantastic feature is the possibility to hide or show the restrictions menu every time the user sets a destination. By letting the users customize their preferences, the navigation in the application will be more pleasant, since the users that never use restrictions don't have to waste time in the respective menu.

The user has available at any time the chance to choose which language he wants to use with the application. The nice thing is that the language is automatically chosen by default as the official language from the country in which the user is located.

Last but not least, the user can add some optional notes about the surroundings of his car when saving its position, as told before. That being said, when the user is in the walking mode after being in the driving mode, he can set his car as destination. The application will route the user to his car's location and will show the saved notes with extra information to make the location process easier.

4. Evaluation

During the evaluation process, it has been tested every possible situation in which TravelBuddy can eventually be used. The tests showed that the application runs smoothly and flawlessly, even on weaker and older android devices, meaning that in each scenario the application was used, the best path was always promptly displayed to the user.

In terms of the information displayed, the most part of the users found that the amount of detail in directions given was appropriate and perfectly clear in each situation.

Since the application was developed from the very beginning taking in consideration the user's feedback, all the requests done are already implemented leading to a great general user satisfaction. The application in its final version is now so simple to use that there wasn't a single user that didn't learn to use it by himself, meaning that TravelBuddy is as user-friendly as it can be.

5. Conclusions

The study of the related work showed that despite the existence of solutions to the presented problem of routing people and vehicles in enclosed spaces, there is no knowledge of a single one that works in every country with every building.

That led to the development of TravelBuddy, which presents itself as an universal solution to be adopted by everyone, since by using low resources of any Android device, it solves the presented problem while quickly giving its users easy and clear instructions of routing in different languages, despite the country in which the users reside or the geographical location of the buildings.

In retrospective it can be seen that the developed application achieved the objectives set from the very beginning for this project, since in addition to guiding people and vehicles in enclosed spaces, it makes it with an extremely intuitive interface that can be easily used by almost any user. TravelBuddy also revealed an algorithm with a good performance since in tests for buildings small to medium-sized (it's difficult to define the point from which a building can be considered large) always showed negligible response times from the point of view of its users.

Despite the fact that no location technology has been implemented, the final version of the application is functional, since the mechanism to use alphanumeric unique identifier for local use as well as the QR codes were shown to be very effective solutions.

Another objective set on early was to be possible to use the same application in any type of space, something that TravelBuddy makes without any problem due to the creation of a language to describe spaces in XML, which allowed to describe all relevant the aspects of any building.

A project like TravelBuddy is never complete, meaning that there is always room for improvement, but in its particular case, that improvement can only be achieved with an expansion of the development team to cover new knowledge areas. So in an ideal future version, TravelBuddy will count with the support for more languages, a newer and better looking interface and finally the support for real-time 3D maps that in addition to the already implemented written orientations helps the user to find his path.

6. References

[1] P. Hart, N. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics,* vol. 4, no. 2, pp. 100-107, #feb# 1968.

[2] R. Dechter and J. Pearl, "Generalized Best-First Search Strategies and the Optimality of A*," *Journal of the Association for Computing Machinery, 32(3):505Â–536,* vol. 32, no. 3, pp. 505-536, 1985.

[3] R. Bellman, "On a Routing Problem," *Quarterly of Applied Mathematics,* vol. 16, pp. 87-90, 1958.

[4] J. Y. Yen, "An Algorithm for Finding Shortest Routes from All Source Nodes to a Given Destination in General Networks," *Quart. Applied Math,* vol. 27, pp. 526-530, 1970.

[5] R. W. Floyd, "Algorithm 97: Shortest path," *Commun. ACM,* vol. 5, no. 6, p. 345+, #jun# 1962.

[6] S. Warshall, "A Theorem on Boolean Matrices," *J. ACM,* vol. 9, no. 1, pp. 11-12, #jan# 1962.

[7] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik,* vol. 1, no. 1, pp. 269-271, #dec# 1959.

[8] D. B. Johnson, "Efficient Algorithms for Shortest Paths in Sparse Networks," *J. ACM,* vol. 24, no. 1, pp. 1-13, #jan# 1977.