HIGH SCHOOL EDITION
Journal of Student Research

How to Use Graph Theory Benefiting the Neighborhoods Environment

Meichen Wan[1] and Matthew Deren[#]

[1]Mentor High School, Mentor, OH, USA
[#]Advisor

ABSTRACT

Graph Theory is a popular field with wide applications. It also has the reputation of being the type of math that is more up-to-date with modern times. However, a quick search of the math curriculum at the secondary level will show a striking absence. This paper hopes to reverse the trend and bring to light both a way to introduce graph theory and also to show the application to the real world all in one unified story. Within this article, the reader will move from little to no knowledge toward a fundamental understanding of one component of graph theory. The reader will also be able to walk away with the ability to analyze efficient routes in real-world networks. This work is a crafted message, a student-to-student message, on building intuition for theories and then using them directly. This article is primarily an argument for the importance of engaging with graph theory early in mathematics study. It argues that graph theory is more than a fashionable buzzword. It is the methodical study of interconnection, and it applies to most of the biggest critical global problems we face today: Environmental Protection, Nuclear Safety, Global Food Supply, and Artificial Intelligence Control. In the end, this article may focus on building one application, resource conservation, and logistic optimization, though the reach will be much further. The tools of graph theory are easily adapted to any problem involving systems and their related parts.

Introduction

In a complex and interconnected world, questions such as the one below appears quite often. Given the map below, what is the most efficient route that returns you to your starting position while traversing every path at least once?

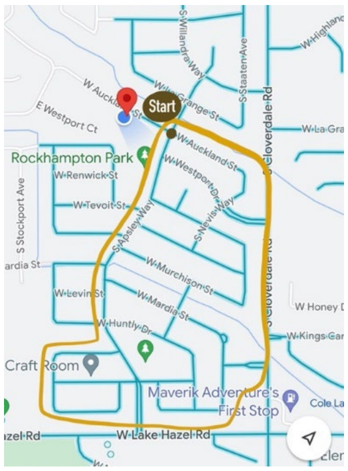


Figure 1. Map of a Neighborhood

This, at first glance, might seem like a difficult and time-consuming problem to solve. However, by the end of this short paper, you will understand this problem clearly and be able to find its solution, and others just like it rather easily.

A natural first approach for a problem of this type might be to use trial and error. In other words, begin by drawing a complete circuit that visits all the paths, and then add up the total distance for that complete circuit. Repeat this process for a new and different circuit and continue until you eventually have every possible circuit and its associated distance recorded. At this point, you can consult your list of records and simply check which path delivers the minimal distance traveled. As you can imagine this technique might only be workable for very simple networks. For larger networks, this trial and error might be highly impractical if not impossible. Given this fact, we might need a more efficient strategy, and as may come as no surprise, there is a field of mathematics that specializes in solving these types of problems. It is commonly referred to by the very broad term "Graph Theory".

Graph theory in its simplest form is the study of relationships within networks. It is normal practice to represent the network system abstractly by points joined together by lines. The graph might be as simple as the one shown below, or as complex as the question presented at the beginning of this article. Either way, we hope to use terms and structures that apply consistently to all graphs. As a point in reference, in graph theory, points are referred to as vertices, and the lines that link those vertices are called edges. So the image below is a very simple example of a graph, which contains three vertices and three edges.

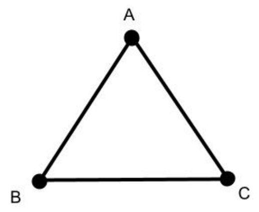


Figure 2.

Using this concept, we can convert our map into an abstract graph that is easier to analyze.

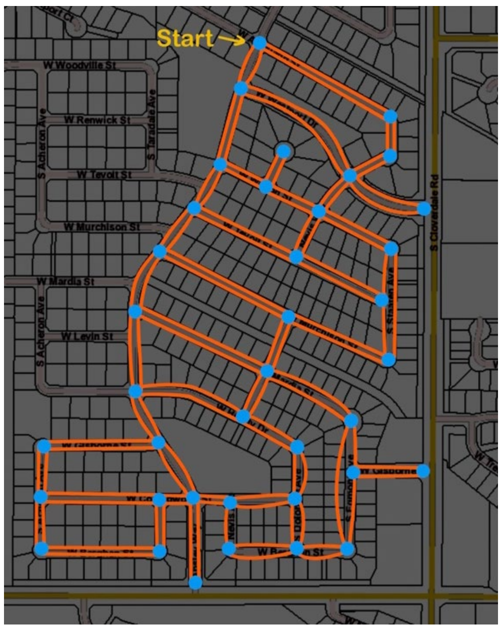


Figure 3. Abstract Graph of the Neighborhood

This front-page problem, and many others just like it, have already been solved by Mathematicians. The first person to categorize this particular version of the problem in modern terms was the Chinese mathematician Mei-Ko Kwan in 1962. It was named the Chinese Postman Problem, referencing an old-fashioned postal delivery clerk that must visit every street in his district (Mei-Ko Kwan, 1962). The essence of the problem asks: how do we find the shortest circuit in a graph that still travels along all edges at least once? In our case, the problem posed at the beginning of this paper is similar to the Chinese Postman problem and is inspired by a model from a small part of my own neighborhood, Rock Hampton II, Boise ID, USA. The solution to the question will be the shortest route for me, as an environmentalist, to clean up the trash on each sidewalk and return to the starting location. The goal is, to beautify our neighborhood with the least amount of walking. While this might seem like a small application, this same process is used in mail delivery, solid waste services, and most logistic delivery industries. In fact, the applications have an even larger reach, as this problem is really about optimal systems in general.

All in all, the problem studied in the paper will help me to change my own surrounding environment, and more importantly, provide an idea of how graph theory can be used for environmental protection.

Section 1. Building the First Concept of Graph Theory

In order to solve our problem, it will be helpful to explore the methods of graph theory using smaller examples. Here is a simple graph that will provide a good foundation for the intuition of graph theory in action.

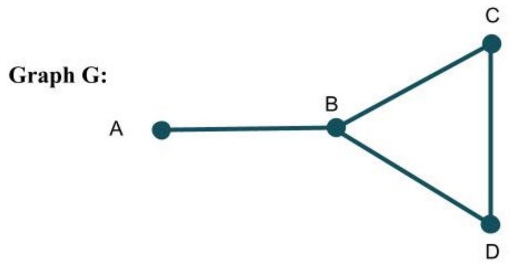


Figure 4. Graph G

In Graph G, we can ask the now-familiar question: what is the shortest route that will satisfy the need to visit each edge at least once, starting from vertex A, and then returning to vertex A?

This graph is simple enough that you can use the trial and error method to see that there are only two routes: A→B→C→D→B→A or A→B→D→C→B→A, and further, they both span the same total distance. In either case, however, we see that edge AB is visited twice with either chosen route. In short, this graph requires repeat travel on an edge.

But let us consider a new graph, Graph G', which is similar but with a very important difference. Here we now have two paths of equal distance connecting vertex A to vertex B.

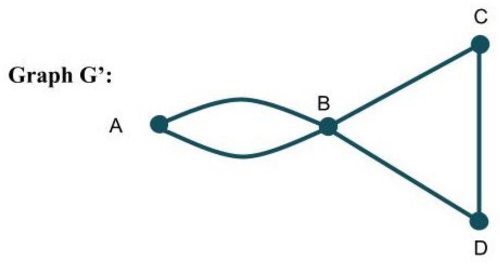
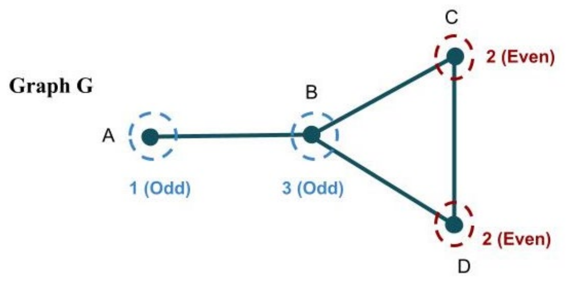


Figure 5. Graph G'

Now with this change, we see that there are four circuit choices, and all of them happen to be of the same total distance. What is different in Graph G', is that we do not need to travel along any edge twice. It is worth pausing here to notice the special property that this graph has. The edges in this graph can be traversed exactly once and this property also happens to be a solution to our problem. Traveling the full circuit without repeating edges is always the best we can ever do!

Graph theory can tell us the way to determine whether the graph has such property. If we look closely at both graphs we see that both Graph G and Graph G' have the same number of vertices, but when we count the number of edges passing each vertex we will see something significant.



Graph 6: Numbers of Edges Connected to Each Vertex in Graph G

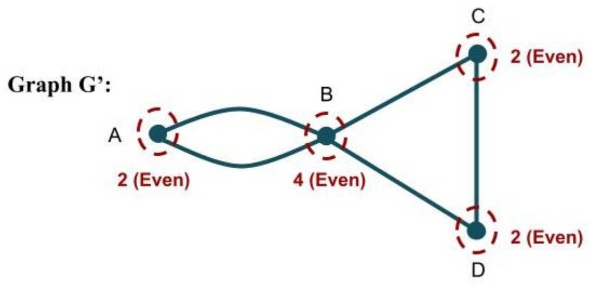


Figure 7. Numbers of Edges Connected to Each Vertex in Graph G'

Notice that the circuit, G', (the one where we do not need to repeat edge travel) has all even vertices, while the graph where repeating edges is required has some odd vertices. From this example, it appears that if a graph has any odd vertices you will have to repeat edges to complete a full circuit. This appearance is actually a true statement and it was first discovered by the mathematician Leonhard Euler in 1736. He stated that only graphs with all its vertices of even degree would contain a full circuit that visits each edge exactly once. This "travel every path exactly once" circuit became later known as an Eulerian circuit. For our part, this means that finding the Eulerian circuit will give us the solution to our opening problem: for we cannot do any better than traveling every path a single time with no repeat journeys.

In terms of graph theory, Carl Hierholzer proved the Eulerian Theorem: A connected graph is Eulerian if and only if each vertex has an even degree (Hierholzer, 1873). This theorem will be useful throughout the paper. It is another benefit of using mathematical structure. We can now take any graph, count the degree of each vertex, and instantly determine if we need to repeat any edge of travel to do a full circuit of the graph.

Section 2: Applying Graph Theory

As we saw in the last section, the Eulerian Circuit becomes the key to solving these types of "visit all edges" problems. If we can determine that the graph has an Eulerian circuit, i.e., the degrees of all the vertices in the graph are even, then we already know that the length of the route that travels each edge at least once will be the sum of the weights of all the edges in the graph. We will still need to find the direction or path's order to complete this circuit and that is where a well-chosen algorithm can help. This will be addressed later on in section 3. However, for now, we will need to address what steps need to be taken if odd degree vertices are found.

If a graph contains any odd degree vertices, then we will need to implement the process of turning Graph G into Graph G' by replicating certain edges to create an Eulerian Graph. As there usually will be many options for which edges to repeat in order to create an all-even vertex graph, we may hope to find which options yield the most efficient circuit. An example will help build the intuition behind these ideas more clearly.

Graph 8 shows that it consists of 4 vertices and 5 paths. The numbers along the edges signify the "weight", which could be cost, time, or in this case the length of the path in meters.

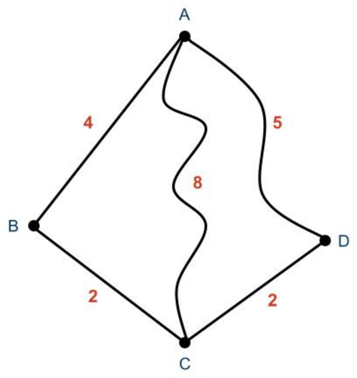


Figure 8.

We don't actually need to show the messy reality of the curves. By using the values of the weights, we can arrive at an easier view with identical information (see Graph H below). With this slight change, we can more abstractly represent this situation and retain accuracy while leaving out unnecessary complications.

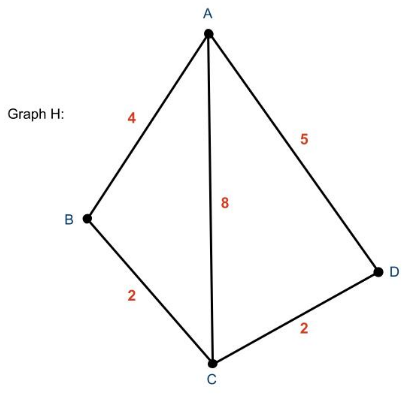


Figure 9. Graph H

So, now that our graph is clearly labeled and displayed, we can proceed. We first determine the degrees of all vertices and notice that our Graph is not an Eulerian Graph: A and C both have degrees of 3. Because there are odd vertices, this graph will require repeating edges to complete a full circuit visiting every edge at least once. In order to convey accurate information and clear records, it is best to mark which edges one plans on repeating during travel. The common practice is to duplicate edges and show them on the original graph, often in a different color. This way one can keep track of what was added in order to convert the original graph into an all-even vertex graph. This we will now do and show the result below.

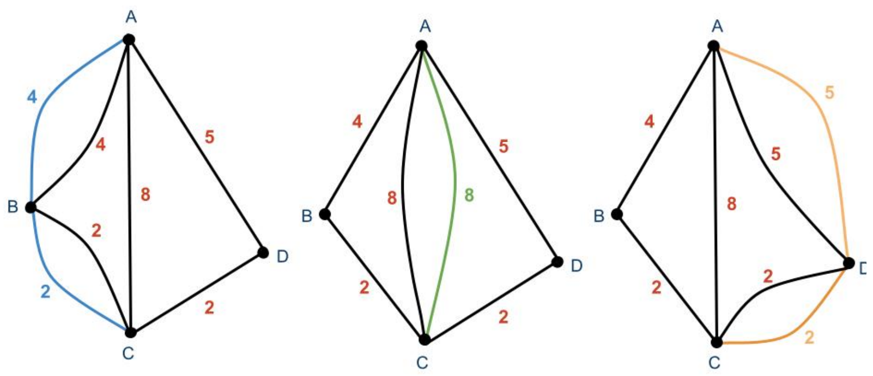


Figure 10. Three ways of duplicating edges

As shown in Graph 10 above, there are three different possible ways of creating a new Eulerian Graph from the original (Non-Eulerian) Graph. So which option should we select? Since there are numerical values assigned to each edge, and if we hope to find the shortest circuit, we could simply add the values together and then choose the repeat edge scenario that creates the minimum total distance. So we can determine that repeating the edges AB and BC will be the most efficient option, requiring only 6 units (whereas the remaining two choices require repeating 7 and 8 units respectively). It is true that selecting the optimal paths to repeat is simple in this small example, but in more complicated cases the best choices may not be so easy to find.

So then, before we go on to a more complicated example, it will be useful to summarize the current knowledge we have developed so far. When we look at a real-world network and wish to answer useful questions about finding an optimal circuit that traverses each edge at least once, we must always accomplish the following:

1. Take a cluttered real-world map and turn it into a mathematical graph with all relevant information of the edges and vertices properly labeled.
2. Count the degrees of every vertex in order to determine whether we already have an Eulerian Circuit appearing in the graph.
3. If our graph does not contain an Eulerian Circuit, we will need to convert it to an Eulerian graph that meets our specific goal, which in our case is the minimum weighted route.

With that concise summary in mind, we can now follow those steps with a slightly more involved example.

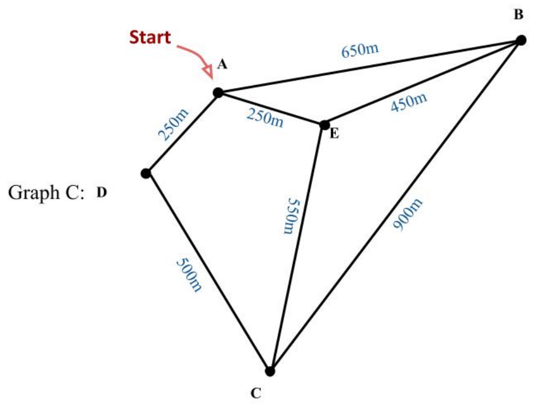


Figure 11. Graph C

Graph C, is an abstract graph converted from a real-world problem (step 1), which shows the area served by a package delivery company. Every morning, a courier will depart from place A, and then will deliver packages to the people living on each road represented by each edge, and then finally return to point A. We now ask: what will be the “minimum weight route” while fitting these requirements?

Step 2: List all vertices with odd degree

Table 12. Degrees of Each Vertex in Graph C

Vertices	Degrees
A	3
B	3
C	3
E	3

From the table above, we see that Graph C is not an Eulerian Graph because it has odd vertices. In this case, we will need to convert all 4 odd vertices to even degree vertices. Unlike the previous graph (with only two odd-degree vertices) this situation has more options for choosing which edges would be the most efficient to duplicate.

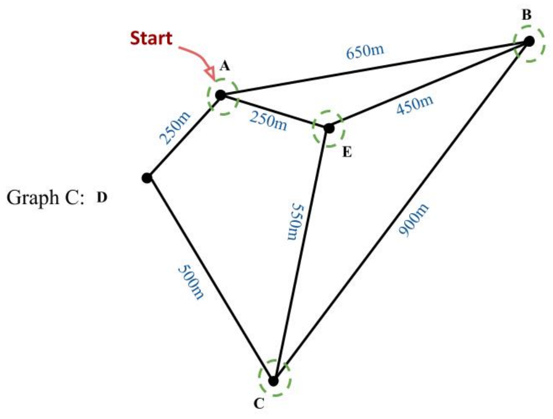


Figure 13. All Odd Vertices Circled

Step 3: Putting odd vertices in groups of two

This step is critical because it is the place where we ensure consideration of all combination possibilities for converting the graph to an Eulerian. Previously, in Graph 10, we had three total possibilities and one of those did in fact give us a minimum path. By replicating all the possible edge combinations that would give us an Eulerian circuit, and analyzing the results, we will be able to solve our problem of minimal travel with absolute certainty. We hope to do the same thing for increasingly complicated graphs with many more vertices and edges.

Fortunately, there is a pattern that can at least tell how many vertex pair combinations we will have to evaluate to make sure we have considered every possible conversion option.

Table 14. Finding Number of Possible Pairs

Number of odd vertices	Number of possible combinations
2	(2-1)

Volume 11 Issue 3 (2022)

4	(4-1)*(4-3)
6	(6-1)*(6-3)*(6-5)
8	(n-1)*(n-3)*(n-5)*(n-7)
10	(n-1)*(n-3)*(n-5)*(n-7)*(n-9)
n	(n-1)*(n-3)*(n-5)*...(n-(n-1))

So for example, with 2 odd vertices, we have one combination to evaluate, with 4 odd vertices, we will need to consider 3 combinations. And the pattern continues, with 6 odd vertices, we will have 15 combinations to consider, and the last entry in the Table shows the pattern for the general case of "n" odd vertices.

You may be wondering why there are no entries for 1, 3, or 5 vertices listed in the table. It turns out that for the type of graphs we are considering (finite and undirected), there cannot be an odd number of odd vertices (Hein, 2015). Why this is the case, can be explored in the book *Introductory Graph Theory* by Gary Chartrand which provides a detailed explanation and proof for this theorem.

To return to our specific example, because our graph has 4 odd vertices, from our formula, we know we will need to analyze 3 different combinations.

Table 15. Combinations

Combination 1	A to B	C to E
Combination 2	A to C	B to E
Combination 3	A to E	B to C

For combination 1, we will have to evaluate all possible edges that link vertices A and B as well as all possible edges that connect vertices C and E, and so on for possibility 2 and 3. We then select the edges with minimum weight in all possible combinations. We will next show the process of selecting the lowest weight for each combination option in the next step.

Step 4: Find the minimum weight combination.

In table 16 we see our first item is A to B. So we search for the minimum way to travel from A to B. The shortest path can be seen to be traveling directly through edge AB (650 meters). By doing the same action for all the remaining table entries, the following table can be completed.

Table 16. Minimum Weight of Edges Between Each Combination

	Minimum weight path	Weight (meters)	Minimum weight path	Weight (meters)	Total weight
Combination 1	A→B	650	C→E	250+250	1150
Combination 2	A→D→C	250+500	B→E	450	1200

Volume 11 Issue 3 (2022)

Combination 3	A→E	250	B→C	900	**<u>1000</u>**

After completing Table 16, we know the best way to convert our graph to an Eulerian Graph. Combination 3 has the lowest weight and so we know that we will choose this combination to create our best Eulerian Graph. The newly transformed graph is shown below with the additional edges displayed in red.

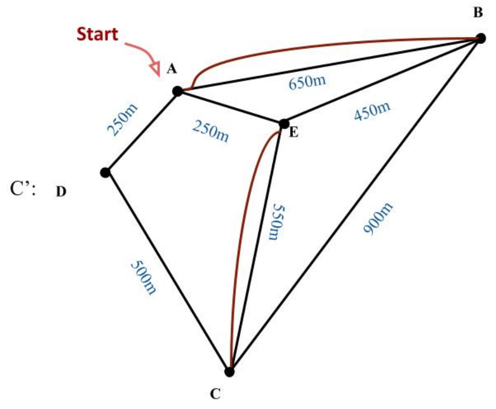


Figure 17. Graph C'

Section 3: Finding the Best Path Travel

So we have our optimal Eulerian Graph but we are not yet completely finished. We still have the task of finding the directions for traveling along the circuit above. For this job, we will use an algorithm to find our path, and there are two well-known algorithms that we can choose from. The first one, Fleury's algorithm, finds the route by simplifying the graph step by step much like crossing out items on a grocery list (Henry, 1883). The second method, the Hierholzer algorithm, breaks graphs into manageable sub-circuits and follows a travel rule for navigation(Hierholzer, 1873). Both of these methods will be explained by working through the examples below.}

Fleury's algorithm:

The basic idea of Fleury's Algorithm is to simplify the graph by removing the traversed edges step by step and keeping an accurate record to detail the order of going through each vertex in completing the circuit. When you encounter multiple choices of travel, Fleury's Algorithm allows for any selection except if the chosen edge is a bridge. A bridge is defined as follows: if removing an edge causes the graph to become unconnected, we call this edge the bridge. A physical example will make the bridge identification much clearer and we will encounter it shortly in our first example.

We begin the algorithm by choosing edge AD as our first path, we delete the edge (keep it recorded somewhere) to indicate that it has been traversed.

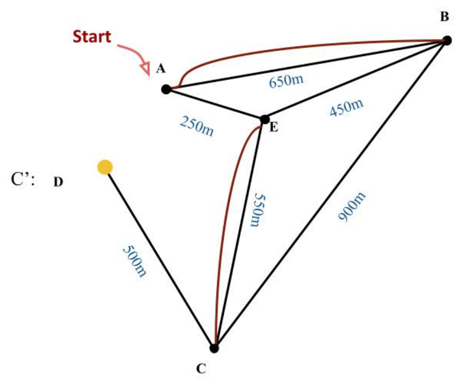


Figure 18. After Traveling Edge AD

Now we are at vertex D, and we see that traversing edge DC is the only option. So we travel along the edge DC and then delete to indicate it has been traversed.

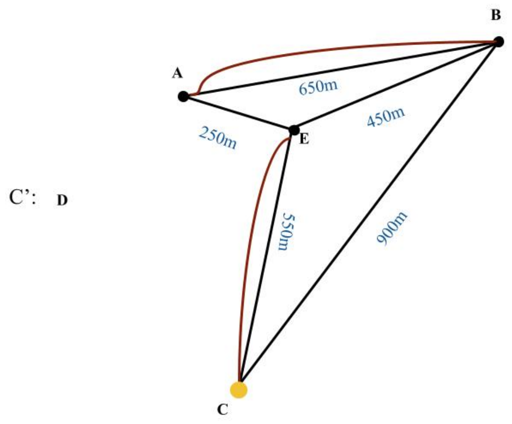


Figure 19. After Traveling Edge DC

At vertex C, there are 3 choices and none of those are bridges. How do we know that these three paths are not bridges? Imagine deleting any one of them and ask yourself does that isolate any part from the rest of the graph? If the answer is no then you know you are not dealing with a bridge. So in this case, we can choose any of the three, and we will choose edge CB.

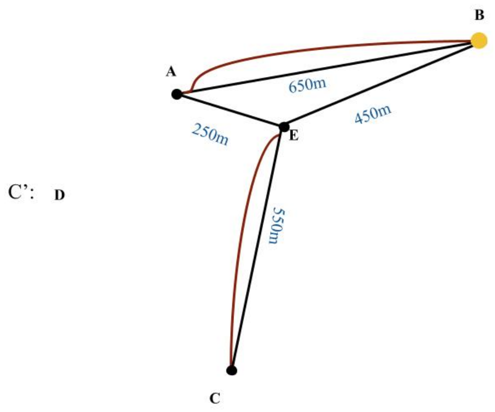


Figure 20. After Traveling Edge CB

This brings us to vertex B, where we have two edges of BA and a single edge BE all three of which are non-bridge edge options we can choose from. So we randomly choose edge BE and remove it as shown in the graph below.

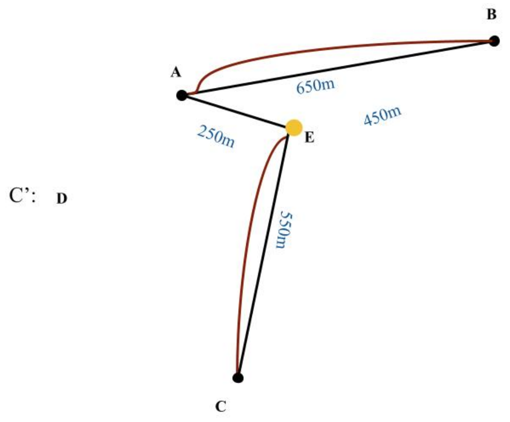


Figure 21. After Traveling Edge BE

At this point we are at vertex E and we are facing another 3 options. However, this time, there is one edge that should not be selected because it is a bridge. If we were to choose edge EA, and then remove it, as shown below, we would see that we have disconnected our graph. This is what we mean by a bridge edge (edge EA is a bridge).

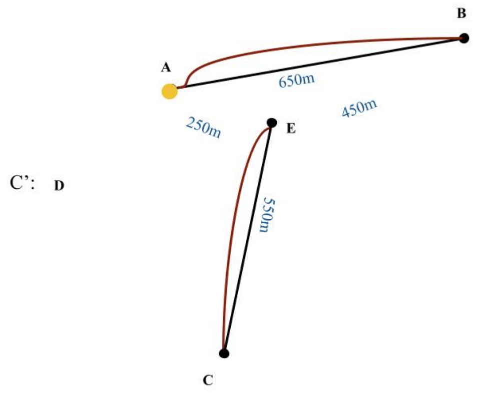


Figure 22. Disconnected Graph After Traversing EA

So, we avoid selecting the bridge and instead choose one of the edges of EC, which brings us to vertex C.

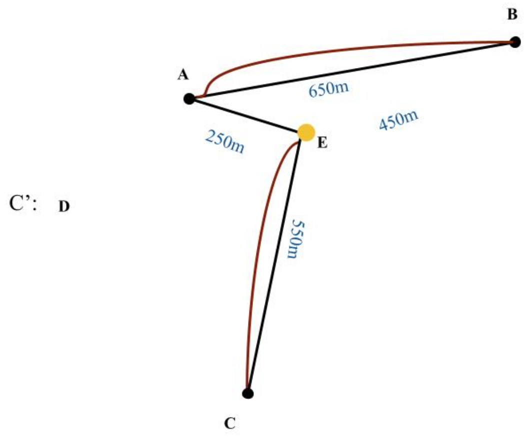


Figure 23. At vertex E

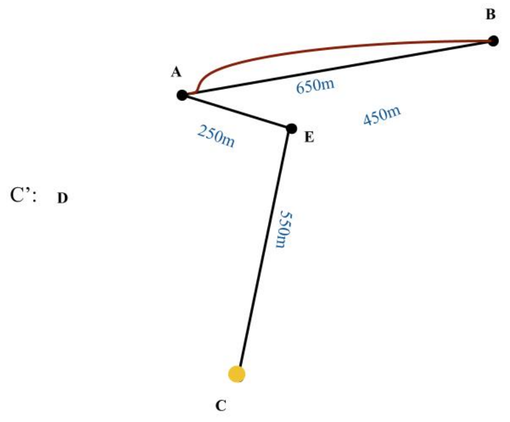


Figure 24. After traveling edge EC

And from C we have only one choice to travel back to E.

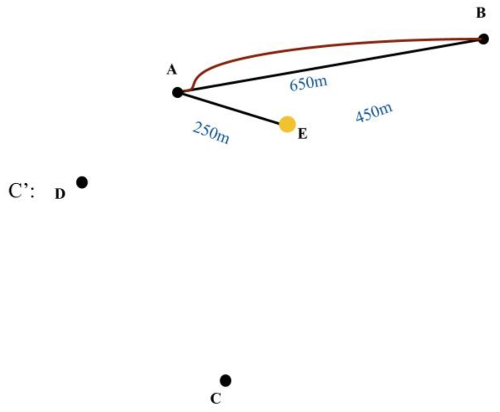


Figure 25. After traveling edge CE

And we next travel from E to A, which at this point is no longer a bridge so we can traverse it.

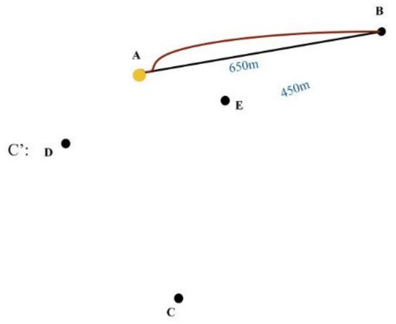


Figure 26. After traveling edge EA

And then we are at vertex A, so we can take either path AB to arrive at B.

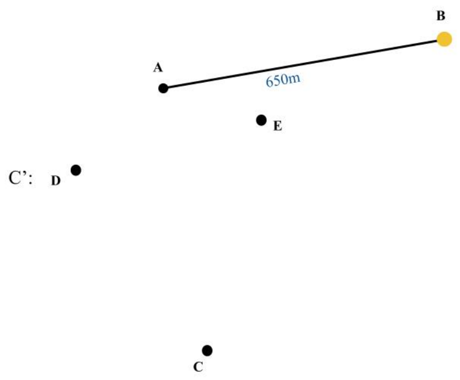


Figure 27. After traveling edge AB

Finally we travel from B to A and complete our circuit: our minimum weight Eulerian circuit path has been determined.

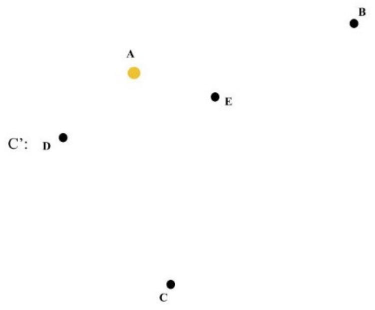


Figure 28. After traveling edge BA

After using Fleury's Algorithm, we can summarize the following steps that we recorded along the way: A→D→C→B→E→C→E→A→B→A. Following this travel will achieve our goal. Now let us use another algorithm to accomplish the same goal.

Hierholzer algorithm:

The first step of this algorithm is to identify the sub-circuits contained in the graph. Starting from vertex A, we randomly choose any circuit: A→D→C→E→A and color it red for record-keeping. This will be our first sub-circuit. Next, we choose any edge, not in the first (red) circuit that still contains a vertex from that first circuit. For example, we could choose edge AB, which meets our requirement, and so we find a new circuit, A→B→A, which we will color in green for record-keeping. This will be our second sub-circuit. Finally, we select edge BE, and find the last circuit B→E→C→B which we color in blue. We now have three distinct connected sub-circuits that compose our full graph.

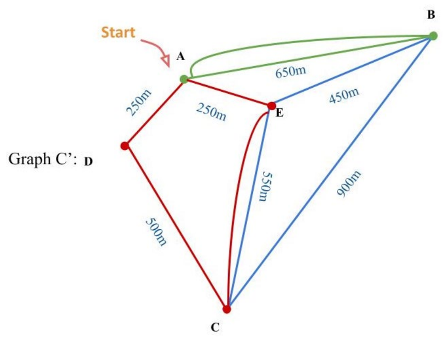


Figure 29. Graph C' with three sub-circuit marked

We are now ready to find a route that will traverse the full Eulerian Circuit. Hierholzer's algorithm sets its rules as follows: you first follow a sub-circuit that contains the starting point (one color) and keep following that sub-circuit in the designated order (determined and recorded above). You follow this path until it joins or meets another sub-circuit (new color). At this point, the rule will be to follow the new color circuit unless part of that color has been partially visited. If this new color has already been traveled in part, you need to stay on your current color circuit. This may sound confusing, but it will become clear as we work through the example.

In figure 29, we have both red and green sub-circuits containing our starting point. So, we could choose either but to begin somewhere we will choose red. Once chosen, we follow our red circuit from A to D and to C. When we get to vertex C, we meet a new color (blue) and that color has not been visited yet; so we proceed to follow the blue circuit. By following the order of the designated blue circuit, we will choose to travel edge CB. When we arrive at vertex B, we encounter another circuit (green). And since green has not yet been visited, we can go through either of the edges of BA and arrive at A. At vertex A, we hit a new color red. However, we have already traversed part of the red circuit, so we will stay within the circuit of our current color (green). Now we are at vertex B, we follow the current blue circuit and get to vertex E. Again, we already traversed part of the red circuit before, so we can stay with our current blue circuit and arrive at vertex C. Now our only choice is to follow the original red circuit, traversing the remaining edge CE and EA. And we are eventually back to our starting point and now have a complete record of how to travel the Eulerian Graph. Here is the record of our travel: A→D→C→B→A→B→E→C→E→A. This is the completion of our goal; this travel plan will be the most efficient route that traverses all the edges in Graph C' only once.

After going through the slow detail of how these algorithms function, we can take a broader view and put these steps into a flowchart to see the big picture. The flowchart begins with any graph and generates the output, an optimized solution path for that graph.

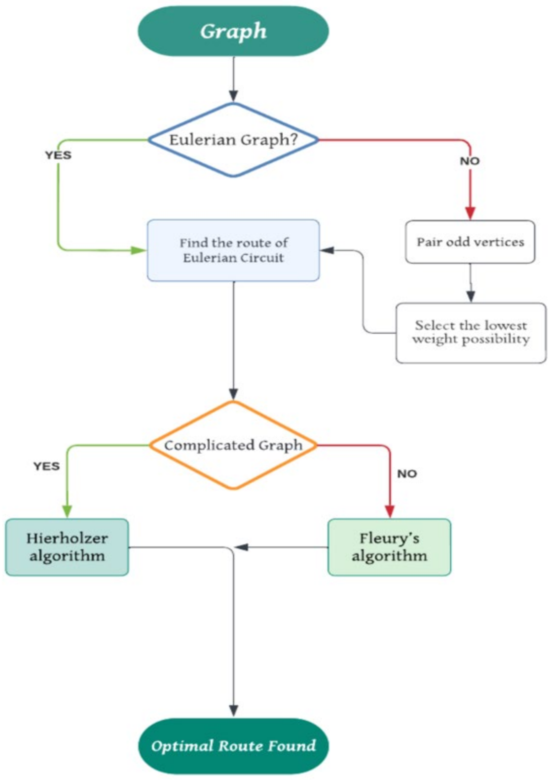
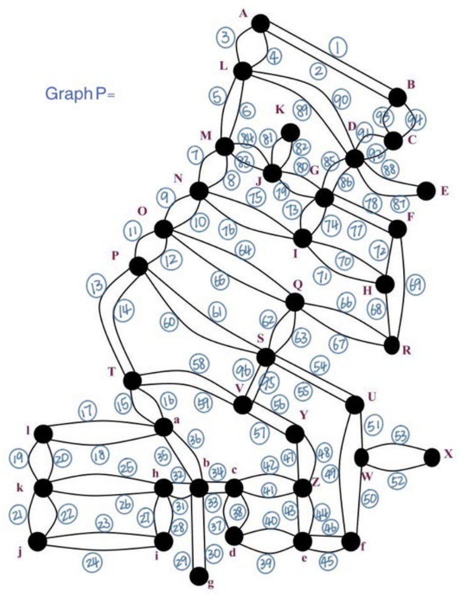


Chart 30. Summary of steps for solving this type of problem

Section 4: Answering Our Original Question

We are now finally in a much better position to solve the problem proposed at the beginning of the paper.



Graph 31. Graph P

Graph P is converted from the real-world map to a clear mathematical graph with 38 vertices and 96 edges. Since every vertex is of even degree, Graph P is an Eulerian Graph. So, fortunately, we now know for certain that we won't need to replicate any edges and we also know the distance of the most efficient route: the sum of all edges. This knowledge, thanks to graph theory, saves us an enormous amount of work and analysis. Figure 28 below, shows the distance measurement tool inside the map in Ada County Assessor, which determines the actual length of the route, which is approximately 11019 meters.

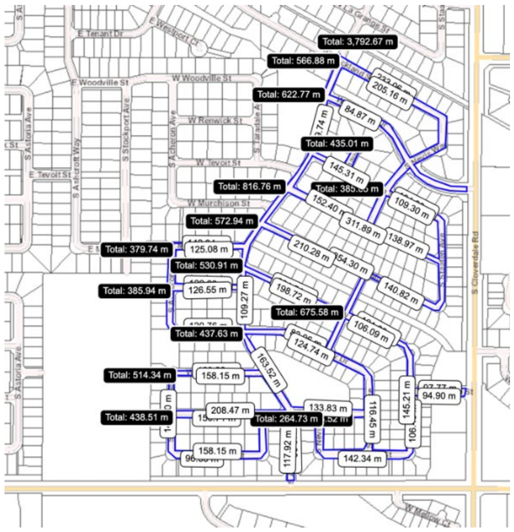
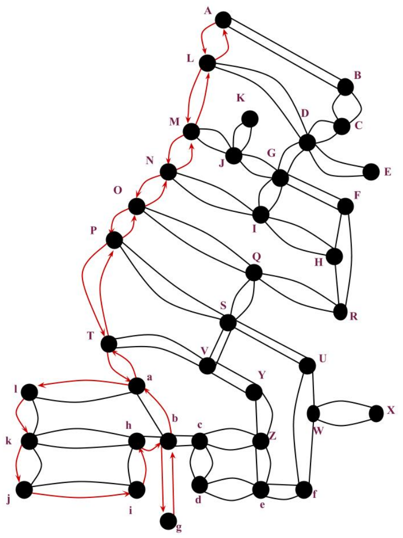


Figure 32. Map of Rockhampton II subdivision

All that remains is the job of finding the specific route to complete this optimal circuit. We already have two algorithms for that. So, which should we use? If we pick Fleury's Algorithm here, it will be time-consuming, inaccurate, and difficult. Choosing Hierholzer's algorithm, where we break a graph into small parts, makes more sense.

Using Hierholzer's algorithm

We will start with vertex A, and then randomly select a circuit starting from this point: A→L→M→N→O→P→T→a→l→k→j→i→h→b→g→b→a→T→P→O→N→M→L→A



Graph 33. Selecting the first circuit

Next, we randomly choose another circuit among the remaining edges as long as it contains at least one vertex from the previous circuit and so on. Because we showed this process in detail already, we will just show the result although the reader can follow along to check the process. Shown in Graph 33 are the other 5 sub-circuits in this graph.

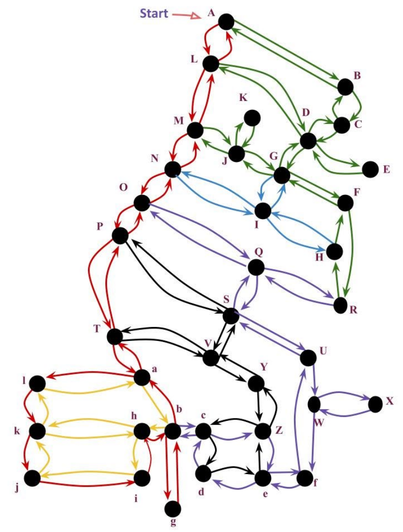
Circuit marked by green, A→B→C→D→E→D→G→F→R→H→F→G→J→M→J→K→J→G→D→L→D→C→B→A

Circuit marked by blue, G→I→N→I→H→I→G

Circuit marked by purple, O→Q→R→Q→S→U→W→X→W→f→e→d→c→b→c→Z→e→f→U→S→Q→O

Circuit marked by black, P→S→V→T→V→Y→Z→c→d→e→Z→Y→V→S→P

Circuit marked by yellow, i→j→k→l→a→b→h→k→h→i



Graph 34. Six selected sub-circuits in Graph P

Our next and final step is to complete our task by finding the instructions for the route of travel. The rules to follow are summarized again for reference below:

- Begin with any circuit that contains the starting point

- Travel in the order of the circuit until you reach a new color
- If the new color has not yet been visited in any part, then move to that new color
- If the new color has been visited in part then stay with your current color circuit

Follow the method above, you will find the solution below:

By following the same steps we did on the previous example, now we can easily determine the most efficient route for Graph P:
A→B→C→D→E→D→G→I→N→M→L→A→L→M→N→O→Q→R→Q→S→P→S→V→T→V→Y→Z→c→d→e→Z→Y→V→S→P→S→U→W→X→W→f→e→d→c→b→c→Z→e→f→U→S→Q→O→P→T→a→b→h→k→h→i→j→k→l→a→l→k→j→i→h→b→g→b→a→T→P→O→N→I→H→I→G→F→R→H→F→G→J→M→J→K→J→G→D→L→D→C→B→A

Following this route, starting at A, is guaranteed to go through each edge once and only once, and end at vertex A. This path will be the optimal route to complete the sidewalk trash collection in a portion of the Rock Hampton II subdivision that we set out to discover! It is important to mention that there will be other circuit orders that yield this same minimum weight route. For example, you can always travel in the reverse direction by reversing the arrow of your discovered circuit. So in a sense, these algorithms provide “a” solution rather than “the” solution.

Conclusion

In this paper, we began with a simple question: What is the most efficient way to travel along all the paths in any given region and return to our starting point? While the question itself seems simple, finding the solution was not necessarily so. The story changed, however, when we used a few simple tools from graph theory. These methods gave us answers to our questions right away: Is it possible to travel every edge only once and return to start? If yes, how do we find the path that manages this feat? And then on the other hand, if we can’t complete the circuit by traveling the edges exactly once, which sides should we repeat in order to minimize our travel? graph theory has the power to transform daunting tasks into a pleasurable journey, but it does much more. It allows us to analyze any network and any system with connected relationships in the real world!

With the viewpoint of graph theory, networks now reveal themselves to be nearly everywhere. Questions of airplane routing, network cable systems, electrical service, wastewater treatments, logistic networks, global climate patterns, molecular interactions in living organisms, and so on. The point is that finding ever more efficient methods to analyze these systems can be the difference between a functioning and improving the world and a world of increasing imbalance. At this point, we can no longer deny that we live in a vast Global Network. It is also safe to say that our biggest problems are global problems, and so, in a way Graph Theory can be a theory that saves the World.

Acknowledgments

I would like to thank my advisor for the valuable insight provided to me on this topic.

References

Mei-Ko Kwan, Programming method using odd or even pints, Acta Mathematica Sinica 10 (1960) 263–266 (in Chinese)

Berge, C. (1964). Graph Theory. The American Mathematical Monthly, 71(5), 471–481. https://doi.org/10.2307/2312582

Lloyd, E. K., Wilson, R. J., Biggs, N. (1986). Graph theory, 1736-1936. United Kingdom: Clarendon Press.
Trudeau, R. J. (2015). Introduction to graph theory. Dover Publications.
Hein, James L. (2015), "Example 3: The Handshaking Problem", Discrete Structures, Logic, and Computability, Jones & Bartlett Publishers, p. 703, ISBN 9781284070408
Chartrand, G. (1985). Introductory graph theory. New York: Dover.
Karinthy, Frigyes. (1929) "Chain Links."