# MATH 400: Network Algorithms

Daniel Loumeau

*Abstract:* Throughout this paper, I will discuss the significance, historical context, and applications of network algorithms. The algorithms discussed are utilized when navigating optimization techniques, modern computing, and urban planning. The evolution of network algorithms is rooted in graph theory. Key concepts include shortest path algorithms, network flow optimization, and routing protocols. In terms of contemporary applications, network algorithms have a tangible impact on many different domains, including telecommunications, transportation logistics, social networking platforms, cloud computing, ...etc. Network algorithms are used in complex environments. This complexity has grown over time due to a multitude of factors. As such, there exist potential trends, emergent challenges, and opportunities that may influence what network algorithms look like in the future.

## 1. *Introduction.*

(1.1) I've taken classes on optimization in the past, and enjoyed applying the mathematical techniques to real-life problems/exercises. Network algorithms have a large role in optimizing and allocating infrastructure within complex systems. So when choosing a topic for my presentation, I considered this a potential subject. After further research, I chose network algorithms due to their relevance in modern technology and the practical mathematics at their source. While I only planned on presenting the basics, network algorithms can range from easy graph traversal to complex optimization problems, making them a great topic for people with different levels of experience. Further, I thought that this choice aligned with the MATH 400 goal of connecting mathematics from different domains. In regards to networks, it can be easily seen how math converges with disciplines including computer science, engineering, and urban planning.

## *2. Historical Context and Development*

(2.1) The evolution of network algorithms is associated with the development of graph theory. Mathematicians like Leonhard Euler, who notably covered the Seven Bridges of Königsberg problem in 1735, made early contributions to graph theory. Another significant milestone for network optimization came from Dijkstra's algorithm in 1959. This method was utilized for finding the shortest path in a graph.

(2.2) In the 20th century, the growth of network algorithms surged even more with the onset of the Internet. As the digital landscape continued to grow and grow, there was a need for controlling online congestion and creating efficient routing paths. As such, network algorithms held more significance and continued to be developed further as environmental complexity increased over time.

## *3. Graph Theory and Algorithm Basics*

(3.1) Graphs are made up of vertices (nodes) and edges that indicate some kind of relationship between them. This magnitude may be distance in time it takes to traverse, physical distance, or some other kind of metric depending on the context. Graphs can be categorized into many different groups, but for the sake of simplicity the major subdivisions are as follows:

- Directed Graphs: Graphs where edges have a given direction demonstrating a specific relationship between nodes. (See Figure 1)
- Weighted Graphs: Graphs where edges have some kind of magnitude between any two given nodes. (See Figure 1)
- Cyclic Graphs: Graphs that contain at least one loop, where it is possible to start at a node and return to that point without backtracking.
- Connected Graphs: Graphs where there is at least one path between every pair of vertices, thus making every node reachable.

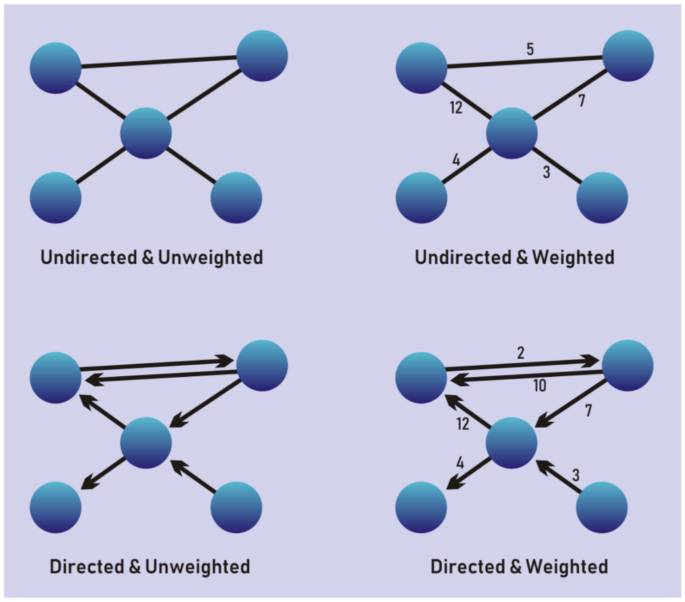

Figure 1.

(3.2) We can label the way algorithms navigate these kinds of graphs into 2 major groups. First, greedy algorithms make locally optimal choices at each step with the hope of finding a global optimum. At each node where the algorithm is to make some kind of decision, a greedy algorithm selects the best possible option without considering future consequences. Because of this behavior, there are naturally pros and cons. Greedy algorithms are usually simple to employ and can be computationally efficient. However, there is no back-tracking backing after you choose a specific route. As such, the global optimum solution is not always guaranteed. Considering this aspect, greedy algorithms are often chosen when finding the optimal solution is less of a priority or when an approximation to the optimal solution is sufficient. Some examples include Dijkstra's, Prims, and Kruskal's Algorithms. The other major group is Non-greedy algorithms. These algorithms evaluate multiple options at each step and consider future consequences before choosing a route or making a decision. With the ability to backtrack, and explore all available options, the optimal solution is guaranteed. Because of this feature, non-greedy algorithms are typically more complex and expensive from a computational standpoint. When accuracy is considered more important for a problem, these algorithms are generally utilized. Some examples include breadth first search and depth first search algorithms.

(3.3) One of the largest graph theory problems is finding the shortest path between two vertices. Dijkstra's algorithm, founded by Dutch computer scientist Edsger W. Dijkstra in 1959, is normally utilized in finding the shortest path in a weighted graph. The algorithm begins at a source node, selects the next node with the minimum distance from it, sets that node as the current node, and continuously updates the distances of its neighbors until all nodes have been visited. The following figure depicts the computational steps to Dijkstra's Algorithm:

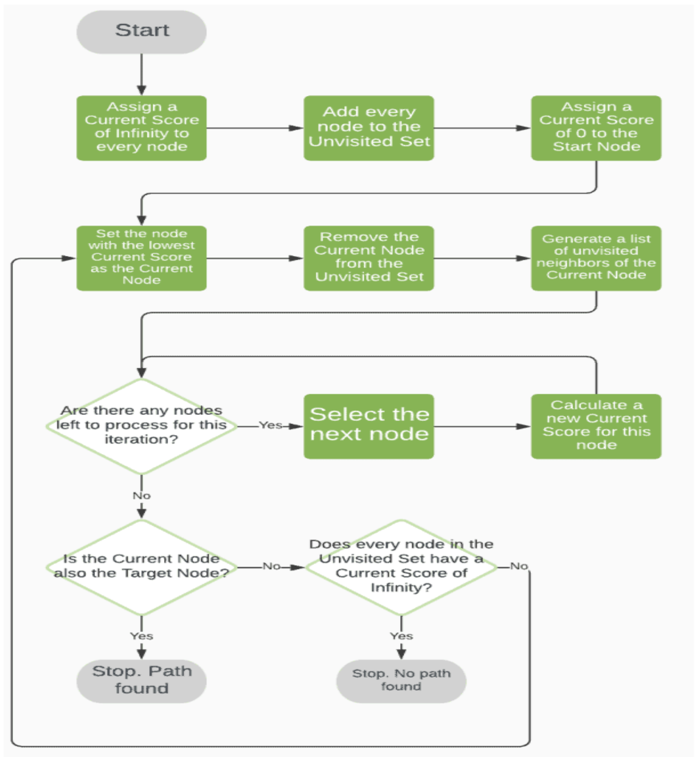
Start
Assign a Current Score of Infinity to every node
Add every node to the Unvisited Set
Assign a Current Score of 0 to the Start Node
Set the node with the lowest Current Score as the Current Node
Remove the Current Node from the Unvisited Set
Generate a list of unvisited neighbors of the Current Node
Are there any nodes left to process for this iteration?
Yes
Select the next node
Calculate a new Current Score for this node
No
Is the Current Node also the Target Node?
No
Does every node in the Unvisited Set have a Current Score of Infinity?
No
Yes
Stop. Path found
Yes
Stop. No path found

Figure 2.

(3.4) Minimum spanning trees (MSTs) are another fundamental component when considering network algorithms. MST's come into play often when dealing with transportation systems. Algorithms that form these tree structures, seek the subset of edges that connect all nodes in a graph with the minimum possible total weight. Two algorithms that do such this are Prim's algorithm and Kruskal's algorithm.

- Prim's Algorithm: This method was initially established by a Czech mathematician named Vojtěch Jarník in 1930. 27 years later, computer scientist Robert C. Prim redeveloped this algorithm in 1957 bearing it a new name. Prim's algorithm starts with an arbitrary vertex and continuously adds the shortest edge that connects a tree node to a non-tree node until all nodes are included. Note a node is only added permitting a cycle won't be formed in the MST by doing so. This algorithm operates efficiently on dense graphs with a large number of edges. The following figure depicts the resulting MST on a simple graph following prim's algorithm starting at arbitrary node 1:

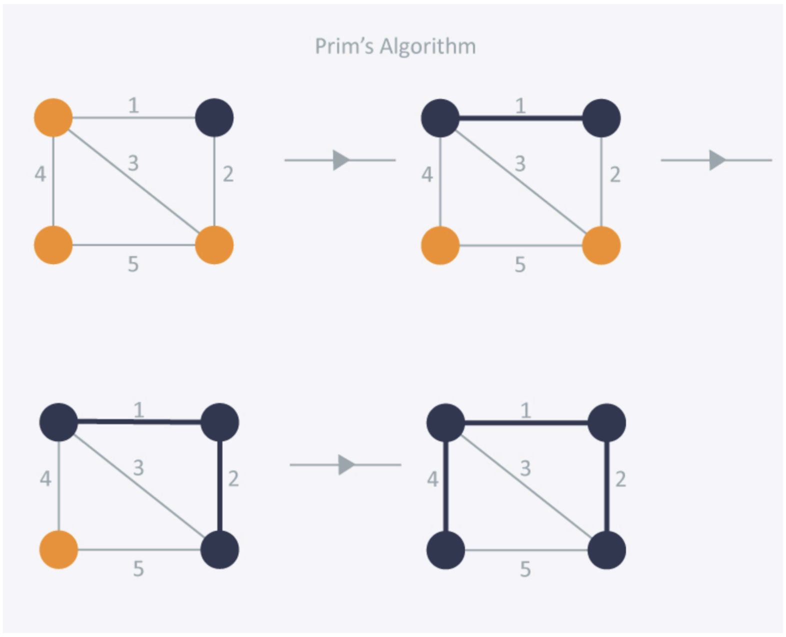

Figure 3.

- Kruskal's Algorithm: This method was introduced by Joseph Kruskal in 1956. Kruskal's algorithm continuously adds the shortest edge to the MST until all nodes are connected, assuming adding an edge won't form a cycle. This algorithm operates in a simple manner and is often utilized on sparse graphs with a few edges. The following figure depicts the resulting MST on a simple graph following kruskal's algorithm starting at arbitrary node 1:

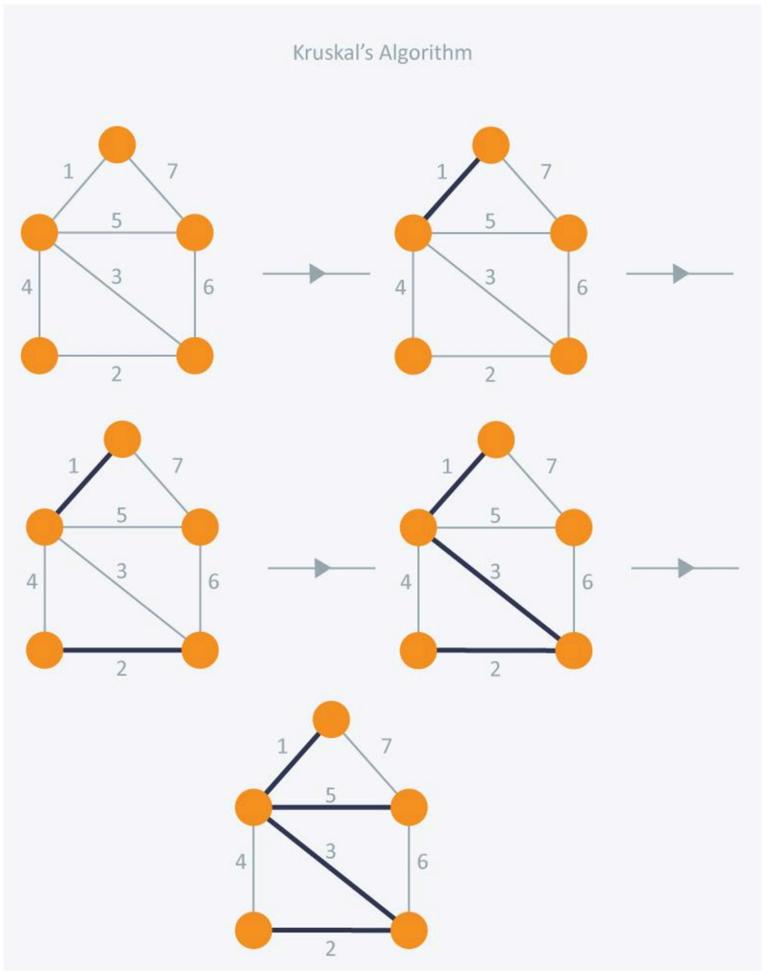

Figure 4.

Understanding how MST's behave under different properties can be vital, as different setups and edge weights can lead to unique MSTs, affecting overall network performance.

(3.5) Breadth-first search and depth-first search are graph traversal algorithms utilized for analyzing graphs.

- Breadth-First Search (BFS): BFS explores a graph level by level. The algorithm begins at an initial node and explores all neighboring nodes before traversing to the next level.
- Depth-First Search (DFS): DFS traverses a route as far as possible along each branch before backtracking. This kind of technique is used commonly for path finding and sorting problems.

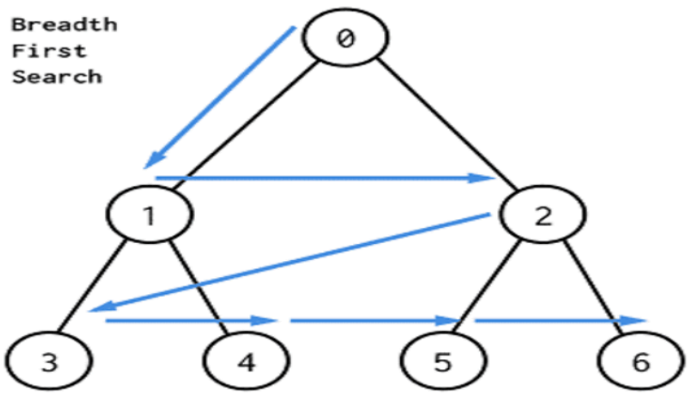

Figure 5.

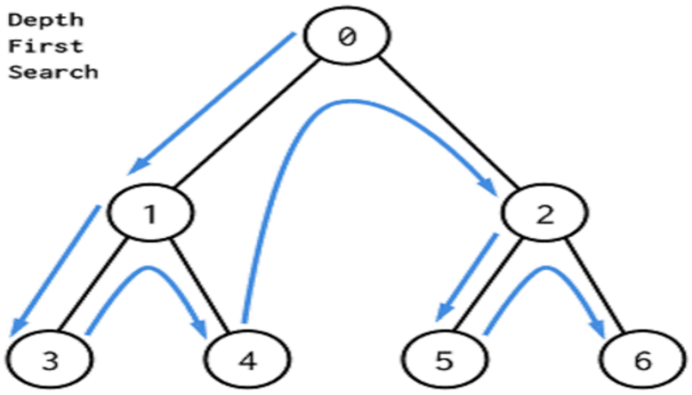

Figure 6.

(3.6) Outside of the traditional algorithms discussed above, network optimization can also be found in nature, with approaches such as ant colony optimization (ACO) and Physarum-Polycephalum simulation.

- Ant Colony Optimization (ACO): As ants search for food, they constantly emit pheromone chemical signals. Over time these scents dissipate. The significance of this is in relation to the distance of a path leading to food. So a long path leading to food, that isn't traversed as frequently, will have a weak scent. On the other hand, a shorter path traversed more frequently, will have a stronger scent. Other ants will recognize this and ultimately abandon the longer paths over time. This behavior inspired ACO, a metaheuristic algorithm with the purpose of solving combinatorial optimization problems. The algorithm was proposed by Marco Dorigo in 1992. It simulates this behavior of ants and is utilized to find optimal solutions to complex problems such as the traveling salesman problem and vehicle routing problem.

- Physarum Polycephalum Simulation: In 2010, researchers from Japan and the U.K. fed a slime mold oat flakes specifically positioned to represent nodes located at the same vertices as that of the Tokyo railway system. From a simple standpoint, the slime mold creates a mesh network that refines over time, strengthening important connections while dropping less important ones. Physarum-Polycephalum replicated a very close model to the already existing Tokyo system. Other researchers have further created models inspired by the behavior of Physarum-Polycephalum to optimize networks. Depending on the context or problem, these models may involve differential equations or other mathematical formulations to include the growth dynamics of the slime mold and incorporate them into optimization algorithms.

All the algorithms discussed above are utilized to create efficient and optimal networks. But there are a lot of factors that can influence how people, resources, or traffic might flow through such networks. So, I briefly wanted to discuss one branch of mathematics that touches this kind of behavior, queuing theory. Queuing theory studies the behavior of queues, or waiting lines, and is necessary for optimizing resource allocation and system performance in complex systems. It involves the study of stochastic processes, probability distributions, and mathematical models to describe the behavior of queues; There are different ways to model the arrival and service of a queue, and understanding how such a process behaves permits a better analysis of a system's waiting times, service capacity, and overall utilization. Different methods may include Little's Law, Erlang's formulas, and various queuing models such as M/M/1, M/M/C, and M/G/1. The M/M/1 queue is a typical format for a basic queue. In this model, we assume there to be a system consisting of a single server. The rate of arrivals is then assumed to follow a Poisson Process, with service having exponential distribution. I wanted to briefly discuss this concept, but I hope to do more research on this topic perhaps in my next presentation.

## *4. Future Advancements/Challenges*

(4.1) Scalability has been a rising challenge for network algorithms due to the exponential growth of network infrastructures. With the potential adaptation of IoT devices (Internet of Things), cloud computing, and edge computing environments, algorithms have to take into account larger volumes of data traffic and more complicated network environments. Dijkstra's algorithm is a good example. It will still work on a given graph but when you're dealing with thousands of potential vertices, the way the algorithm is structured may make it inefficient. A* is an additional algorithm based on Dijkstra’s that Google maps uses to counter this scalability issue. Essentially it incorporates the distance from the start node to the current node in addition to the estimated cost from the current node to the end node. Hence, it’s more efficient in finding the ideal path when dealing with many scenarios.

(4.2) The adaptation of machine learning (ML) and artificial intelligence (AI) techniques also presents opportunities for further developing how network algorithms are utilized. With such a toolkit, future networks may exhibit self-learning and self-optimization. I primarily focused on urban network design and traffic flow for this presentation, but by leveraging ML/AI techniques for networks in any kind of field, network algorithms can respond to changing network conditions, identify incoming threats, and optimize network performance all in real time. Another concern regarding transportation and network layouts is climate change/environment sustainability. Future network algorithms may prioritize solutions that minimize carbon emissions, promote energy-efficient transportation modes, and incorporate green infrastructure into urban planning initiatives. Future network algorithms may also need to support multi-modal transportation systems, including not only cars but also public transit, bicycles, pedestrians, and emerging modes such as electric scooters and drones. It should be noted that these situations are all potential concerns, so the likelihood that such situations materialize is not 100 percent guaranteed. Regardless, they are still important considerations and issues to keep in mind when discussing the future of networks.

## *5. Conclusion/Reflection*

(5.1) As technology continues to develop, network algorithms will remain at the center of innovation. This paper has discussed the historical context, practical applications, and future prospects of these algorithms. Their impact on society will continue to be great considering the role they have in  shaping both digital and physical landscapes. I've enjoyed doing research on this topic and presenting in front of my peers. My classmates have given me positive comments as well as constructive feedback regarding how best to explain this material and where to go from here. There were suggestions about edge computing and its relation to networks, going deeper into the nature aspect, expanding on more of the historical context, and specifically analyzing when and where to use algorithms like prims or kruskal's depending on the context. I've enjoyed learning from my classmates and plan on incorporating this feedback when planning for my next presentation.

*References*

- *Leonard Euler's solution to the Konigsberg Bridge problem*. Leonard Euler's Solution to the Konigsberg Bridge Problem | Mathematical Association of America. (n.d.). https://maa.org/press/periodicals/convergence/leonard-eulers-solution-to-the-konigsberg-bridge-problem#:~:text=this%20famous%20problem.-,Euler's%20Proof,and%20any%20number%20of%20bridges.
- *Ant colony optimization*. Ant Colony Optimization - an overview | ScienceDirect Topics. (n.d.). https://www.sciencedirect.com/topics/engineering/ant-colony-optimization
- ScienceDaily. (2010, January 22). *Slime design mimics Tokyo's rail system: Efficient methods of a slime mold could inform human engineers*. ScienceDaily. https://www.sciencedaily.com/releases/2010/01/100121141051.htm
- Singh, S. (2024, March 7). *The algorithms behind the working of Google Maps*. Medium. https://medium.com/@sachin.singh.professional/the-algorithms-behind-the-working-of-google-maps-73c379bcc9b9#:~:text=Google%20Maps%20essentially%20uses%20two,defined%20by%20edges%20and%20vertices.
- Chris, K. (2023, February 14). *Prim's algorithm – explained with a pseudocode example*. freeCodeCamp.org. https://www.freecodecamp.org/news/prims-algorithm-explained-with-pseudocode/
- Srivastava, A. (2022, April 6). *Kruskal's algorithm*. Medium. https://srivastava1703.medium.com/kruskals-algorithm-800eefc7c9e1
- T&scaron;ernov, K. (2022, February 21). *The beginner's guide to queuing theory*. Qminder. https://www.qminder.com/blog/queue-management/queuing-theory-guide/
- &#27946;&#20581;&#32724; Hung, C. (2022, September 22). *Is BFS/DFS a greedy algorithm? what's the difference between greedy and heuristic algorithm?* Medium. https://hungchienhsiang.medium.com/is-bfs-dfs-a-greedy-algorithm-whats-the-difference-between-greedy-and-heuristic-algorithm-8b6b019c43c1#:~:text=The%20term%20%E2%80%9Cgreedy%20algorithm%E2%80%9D%20refers,it%20to%20an%20optimization%20problem.
- Bettilyon, T. E. (2019, May 7). *Types of graphs*. Medium. https://medium.com/tebs-lab/types-of-graphs-7f3891303ea8
- Gaskins, A. (2020, November 19). *Distinguishing BFS and DFS*. Medium. https://ashley-gaskins.medium.com/distinguishing-bfs-and-dfs-bb413fa1b0e7
- Gaskins, A. (2020, November 19). *Distinguishing BFS and DFS*. Medium. https://ashley-gaskins.medium.com/distinguishing-bfs-and-dfs-bb413fa1b0e7
- *Minimum spanning tree tutorials & notes: Algorithms*. HackerEarth. (n.d.). https://www.hackerearth.com/practice/algorithms/graphs/minimum-spanning-tree/tutorial/