Networks and Graphs

Math 499H
Dr. Rodine

Brenda K. Tsao
April 1985

1. Introduction

Graph theory is a part of mathematics that has many practical applications. The study of graph theory began in 1736 with Leonard Euler's work on the Konigsberg bridge problem. Konigsberg was a city built on a river with two islands and seven bridges. The question was posed as to whether you could start anywhere, cross each bridge exactly once and end up at the starting point. Other problems that arise include scheduling routes for transportation systems that meet certain requirements such as distance, time, places stopped, and profits; and choosing the best times for replacement of equipment and the best type to use -- what brand, new or used -- to make operations most efficient.

The focus of this paper will be on algorithms used for optimizing problems. It will cover the uses of these algorithms and why they are used as well as touching on how each one works.

2. Tree Algorithms

Tree algorithms can be divided into two groups: spanning tree algorithms and maximum branching algorithms. Spanning tree algorithms are used when the object of the problem is to get all the vertices of a graph connected; for example, a company having several buildings must construct passageways between them, with no constraints on how they connect the buildings. A problem of possible interest to the company may be how to connect the buildings with sidewalks using the least amount of pavement.

Spanning Trees

The spanning tree algorithm examines each edge of the graph to determine if it is needed to connect the vertices. If it adds a new vertex to the graph, it is kept; if it is redundant, it is not kept. The algorithm stops when all the vertices in the graph are connected or there are no more edges to be considered. The algorithm is easily adapted to find the maximum and minimum spanning trees by examining each edge in descending or ascending order respectively, rather than arbitrarily examining any edge.

This algorithm is very efficient, because it examines each edge at most once, and may stop before examining all of the edges. It is also a very easy algorithm to implement. It requires only one decision per edge, with the arcs being sorted by value for a maximizing or minimizing problem.

Maximum Branching Algorithm

Maximum branching algorithms are used in cases where the direction of movement along an edge is to be considered. When the edge is directed it is called an arc. A special kind of spanning tree is called an arborescence. In an arborescence each vertex has at most one arc going into it, but it may have several coming out. An example of this would be the chain of command in a company. The president is in charge of the divisional vice presidents, who each send orders to their managers, who tell their workers what to do.

The general maximum branching algorithm examines each vertex in an arbitrary order. For each vertex, the arc of maximum value that enters it is added to the graph, if such an arc exists. This process continues until there is no longer a branching. When a circuit is formed, it is considered a single unit, given a name, and a new weight is calculated for it. This process continues until all of the vertices in the original graph have been considered. Next the circuits are expanded to their original state and the appropriate arc is deleted from each circuit to make it a branching. This maximum branching will also be the maximum spanning arborescence, if one exists. Minimum branchings are found by using the negative of each arc weight and following the same algorithm.

To root a maximum spanning arborescence at a specific vertex, a false vertex is created, and one arc is added from this vertex to the root vertex. The value of this arc is arbitrary. Since no arcs enter the false vertex, the arborescence, if it exists, will be rooted there. After performing the algorithm, simply disregard the false vertex and its arc.

Because of the directional requirement, this algorithm requires more computation. Although this algorithm requires reexamining the vertices in a circuit, it is efficient because it adds vertices one at a time until they have all been considered. Once added, a vertex is never removed from the tree.

Tree algorithms are based on the same structure as a tree in that the whole thing must be connected, but no branch splits into parts and later rejoins any of the branches from which it split. By consistently taking the largest possible arc and maintaining a branching, a maximum branching is ensured. This will span the graph if possible; anything less would not be a maximum.

3. Shortest Path Algorithms

Shortest path algorithms are used in problems where there are several alternatives, and the object is to choose the one requiring the least amount of time or expense. For example, if a person wanted to drive from Chicago to DeKalb, what would be the shortest route? A more generalized form would be the Kth shortest route. For example, if the traveler wanted to return to Chicago by bus, he may want to find the fastest route. Also, in case there is no more room on that trip, he would also like to know which is the second fastest route.

Dijkstra Algorithm

The Dijkstra algorithm begins at the initial vertex and chooses the shortest path from there to each of the other vertices, the shortest of which it connects. Then it continually takes the vertex nearest the initial vertex and connects it until the final vertex is connected. In choosing the shortest path, the Dijkstra algorithm considers both the distance directly from the initial vertex and the distance from the initial vertex through other connected vertices. This finds the shortest path from the initial vertex to the final vertex.

This algorithm is efficient in that one vertex is added at each iteration until the shortest path is found, but it requires calculating the distance for each vertex until it is connected. The Dijkstra algorithm must be modified if any of the arc lengths are negative. The Ford algorithm does this by reconsidering the connected vertices each time, and changing the shortest distance if one is found, but this requires more calculations and more steps.

Floyd Algorithm

The Floyd algorithm begins by setting up an initial matrix whose elements are the lengths of the shortest arc between each pair of vertices in the graph. Each iteration consists of choosing the minimum distance from: the previous matrix or the first vertex to the second via a given vertex. These values make up the new matrix. The process is then repeated until every vertex has been considered as an intermediate vertex.

The Floyd algorithm is advantageous in that it calculates the shortest path from each vertex to every other vertex in the graph, whereas the Dijkstra algorithm finds the shortest path from one given vertex to another given vertex. Because of its matrix format this algorithm is easier to implement on a computer than a graph format algorithm such as Dijkstra's. The Dantzig algorithm also has a matrix format.

Dantzig Algorithm

The Dantzig algorithm works much like the Floyd algorithm except that in the Floyd algorithm the shortest path can be modified in each iteration when a new vertex is considered as an intermediate vertex. In the Dantzig algorithm the initial matrix is a one by one matrix. In each iteration a row and a column are added to the matrix and the shortest path through any intermediate vertex is considered. In this case, the shortest paths are not modified. Essentially the Floyd and Dantzig algorithms require the same operations in a different order.

Kth Shortest Paths

Kth shortest routes, as mentioned previously, are often of interest as alternatives. One method of finding the Kth shortest path is the Double Sweep algorithm, which finds the Kth shortest path from any given vertex to every other vertex in the graph. The Generalized Floyd and the Generalized Dantzig algorithms also find the Kth shortest paths. For these algorithms a vector is substituted for each element in the matrix, and a generalized minimization and addition are defined and used. These algorithms solve for the shortest through the Kth shortest paths from each vertex to every other vertex in the graph.

The main advantage of the generalized Floyd and Dantzig algorithms is that they solve for the Kth shortest path between all pairs of vertices. Except when the Kth shortest path from only one destination is of interest, the double sweep algorithm is less efficient, because it must be repeated once for each additional initial vertex of interest.

While shortest path problems can be solved using a matrix format, the idea of the shortest path problem is much like a map. It shows many alternative routes from one destination to another, and from them the best route must be chosen. The path need not be direct, but it must follow existing roads.

4. Flow Algorithms

Flow algorithms can be considered as two major types of problems. The first is when the amount of flow through a network needs to be maximized. The second is when a set amount of flow must be moved through a network, and the object is to minimize the cost.

Maximum Flow Algorithm

The maximum flow algorithm begins with a network in which each arc has a given capacity. The first step is to choose any path from the source, or beginning point, to the sink, or final destination, and send as many units of flow through that path as possible without violating a capacity. Each successive new path found may consist of any unused flow capacity in the direction of the arc, and any used flow capacity in the opposite direction. When all possible paths are found, any time a backwards flow was used, two paths were crossed. The crossed segments are removed, and the beginning of each path is connected to the end of the other. This algorithm can be modified to accomodate several sources and sinks. This is done by connecting a "false" vertex with infinite arc capacity to each source, and a second "false" vertex to each sink.

Minimum Cost Flow Algorithm

The minimum cost flow algorithm is useful for problems in which a given amount of material must be shipped from source to sink over routes that may have limited capacities. The object is to ship the goods at a minimal cost. This algorithm begins by giving each vertex an initial value of zero. It then operates in the same way as the maximum flow algorithm with the additional constraint that the difference between the numbers assigned to two vertices must equal the arc cost. In the next step, the value of each vertex that has not yet been added is increased by one. By repeating this process, the algorithm finds first, all paths with cost zero, then cost one, cost two, and so forth until either the given number of flows has been found or the maximum flow is reached. Since non-zero bounds are not permitted in this algorithm, another algorithm may sometimes be needed.

Out-Of-Kilter Algorithm

The Out-Of-Kilter algorithm is used when non-zero lower bounds exist as long as there are no negative circuits of infinite capacity. The algorithm begins by choosing an arbitrary set of flows from source to sink. For each arc, a number based on the principle of complementary slackness, and the difference between the actual flow and the minimum and maximum capacities must be calculated. From these figures a kilter number, or measure of how much the situation is out of kilter, is assigned to each arc. For each out of kilter arc, the algorithm searches for a flow that will improve the kilter situation without worsening the kilter situation of any other arc. This is repeated until no more flows can be sent from source to sink. Then the vertices are renumbered and the process is repeated until everything is in kilter, or no feasible flow exists.

Dynamic Flow Algorithms

Dynamic flow algorithms are used when there is a time constraint in a flow problem. Besides having a flow capacity, the time to travel from one vertex to another along each arc is known. Given a constraint of n units of time, these problems can be expanded by expanding each vertex into n vertices to represent the units of time. Corresponding arcs are drawn from one vertex to another moving to the appropriate time spot on the vertex.

The new graph can be solved with standard flow algorithms. Expanded graphs do not require new algorithms, but they can become very inefficient as the graph gets bigger with each additional vertex and longer time constraint.

Maximum Dynamic Flow Algorithm

The maximum dynamic flow algorithm utilizes the minimum cost flow algorithm by using the travel time of the arc as the cost. The final flow from this algorithm is broken down into paths, and a new flow is sent down each path for each unit of time until a flow sent would not reach the sink before the time constraint was reached.

Earliest Arrival Flow Algorithm

Another type of dynamic flow concerns the earliest and latest departure and arrival times. The earliest arrival flow algorithm also uses the minimum cost flow algorithm. It works in the same way as the maximum dynamic flow algorithm with the additional condition that earlier arrivals are done first; that is, the algorithm first considers flows with arrival time zero, then those with arrival time one, and so forth, until the given number of flows or the maximum flow is reached. Latest departure flows can also be calculated by reversing the arcs in the graph and using the earliest arrival flow algorithm. Latest arrivals, unfortunately, are much more complicated, because they involve holdovers at vertices along the path.

Flows With Gains Algorithm

Finally, another type of flow problem is encountered when the amount of flows are allowed to be changed along the way. An example of this is an investment problem, where an investment may acquire either a gain or loss along a particualr arc, and the new amount must be invested in the next arc along the path.

In these types of problems, each arc in the graph has a gain factor. If all gain factors are one, the problem can be solved with the out-of-kilter algorithm. If the problem contains absorbing or generating cycles, where losses or gains occur, the flows with gains algorithm must be used. This algorithm begins by finding a set of flow values and dual values that form a feasible solution in which there is complementary slackness. This set may contain fewer than the required number of flows to be dispatched from the source. The algorithm then searches for additional flows that will still satisfy the original conditions. When this is no longer possible, the dual values are then altered to stay within the conditions and allow more flows to go through the network. The algorithm continues to alternate between these two steps until the given number of flows have been found, or the maximum number of flows has been reached.

Although flow problems are more diverse in their variations than shortest path problems, they also work much like a map. But in the case of flow problems, capacity rather than distance is the main factor. The algorithms look for many paths through the network rather than the shortest path.

5. Conclusion

Graph theory is useful in many types of problems in many different fields. Often it is used for linear programming problems where algorithms such as the simplex algorithm will work, but are not as efficient. The algorithm used to solve such a problem depends on the type of problem. In some situations, more than one algorithm is available. In these cases the choice may depend on the conditions of the problem, which may make one algorithm more efficient than another.

Bibliography

Hillier, Frederick S. and Lieberman, Greald J. Introduction to Operations Research, Third Edition. Oakland, CA: Holden Day, Inc. 1980.

Minieka, Edward. Optimization Algorithms for Networks and Graphs. New York, New York: Marcel Dekker, Inc. 1978.