Network Flow Problems

Paul F. Roysdon, Ph.D.

I. Introduction

Contemporary machine learning applications attempt to solve network flow (netflow) problems from a variety of mathematical approaches. Herein we take the classical approach applying Linear Programming (LP) to netflow and consider the following:

- Shortest route problem.
- Maximum flow problem.
- Minimum cut problem.

We formulate each problem from the min-cost LP perspective.

II. Shortest Route Formulation

We begin this study with a simple minimization problem, and refer the reader to "Flows" in Chapter 6 of [1], and "Digraphs" of Chapter 4 in [2].

A. Directed Graphs

Using the notation in Chapter 1 of [1], a directed graph (or digraph) is a set of vertices, $v \in \mathbf{V}$ and a set of directed edges, $e \in \mathbf{E}$ that each connect an ordered pair of vertices. We say that a directed edge points from the first vertex in the pair and points to the second vertex in the pair. We use the labels 1 through m for the vertices in a $\mathbf{V}$-vertex graph.

Sources, destinations and intermediate points are collectively called *vertices*, while links connecting the nodes are termed *edges*.

Define x_i as the unknown flow in the edges. Then *flow conservation at the vertices* is defined as

$$\sum_{out} x_i - \sum_{in} x_i = b_i$$

Consider the digraph in Figure 1 with one source v_1 and sinks v_{11} and v_{11}. The arrows indicate the direction for each edge from one vertex to another vertex.

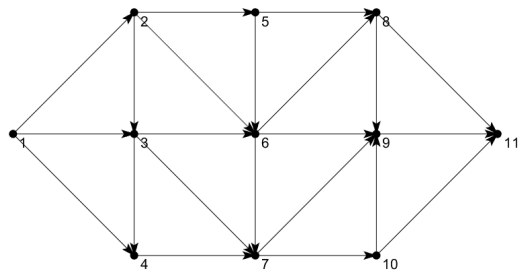

Fig. 1. The digraph with annotated vertices.

B. Shortest Paths

The shortest path minimizes the weights (lengths) of the edges. To find the shortest path between all $v \in \mathbf{V}$ for a weighted graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$.

Let a simple path η be defined as $\eta = \{v_1, v_2, \ldots, v_m\}$. An *intermediate vertex* is any vertex $\{v_2, \ldots, v_{m-1}\}$.

Define w_{ij} the weight of an edge between vertex i and vertex j in graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, where

$$w_{ij} = \begin{cases} 0 & \text{if } i = j, \\ \text{weight of edge } (i,j) & \text{if } i \neq j \text{ and } (i,j) \in E, \\ \infty & \text{if } i \neq j \text{ and } (i,j) \notin E. \end{cases}$$

Finally, $\mathbf{W} \in \mathbb{R}^{m \times n}$ matrix of edge weights of an m-vertex graph, where $\mathbf{W} = [w_{ij}]$.

Define $d_{ij}^{(k)} \in {}^{+}\mathbb{R}^{m}$ as the weight of the shortest path from vertex i to vertex j for which all intermediate vertices are in the set $\{1, 2, \ldots, m\}$. Let $D^{(k)} \in {}^{+}\mathbb{R}^{m}$ represent the path distances between vertices in a directed m-vertex graph, where $D^{(k)} = (d_{ij}^{(k)})$.

Observations:

- From the definition of D, the shortest path cannot have the same vertex more than once.
- For a shortest path from i to j such that any intermediate vertices on the path are chosen from the set $\{1, 2, \ldots, k\}$, there are two possibilities:
  1) k is not a vertex on the path, so the shortest path has length $d_{ij}^{(k-1)}$.
  2) k is a vertex on the path, so the shortest path is $d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$.

Therefore we can recursively define $d_{ij}^{(k)}$ as

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0, \\ \min\left(d_{ij}^{(k-1)}, \;\; d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \geq 1. \end{cases} \quad (1)$$

To improve the search, we can impose a condition on the predecessor of each vertex in the path, such that the predecessor of vertex j on a shortest path from vertex i with all intermediate vertices is

$$\pi_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j \text{ or } w_{ij} = \infty, \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty, \end{cases} \quad (2)$$

and

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, \\ \pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}. \end{cases} \quad (3)$$

Then the predecessor matrix is $\Pi^{(k)} = \left(\pi_{ij}^{(k)}\right)$.

C. Floyd-Warshall Algorithm

From eqns. 1, 2 and 3, the Floyd-Warshall algorithm is defined in Algorithm 1, with runtime $\mathcal{O}(n^3)$ and space $\mathcal{O}(n^2)$.

Algorithm 1: Floyd-Warshall Algorithm.

Initialization: $n = \text{rows}(\mathbf{W})$, $D^{(0)} = \mathbf{W}$, $\Pi = \mathbf{0}$;
for k 1 *to* n **do**
  for i 1 *to* n **do**
    for j 1 *to* n **do**
      • if $d_{ij} > d_{ik} + d_{kj}$;
      • then $d_{ij} = d_{ik} + d_{kj}$;
        $\pi_{ij} = \pi_{kj}$;
    end
  end
end
Result: Shortest path between vertex 1 and vertex n.

D. Example: Minimum Edge Weight

Using the graph shown in Fig. 1, we can assign edge weights as shown in Fig. 2 and solve the shortest path problem using Algorithm 1. The solution is shown in Fig. 3.

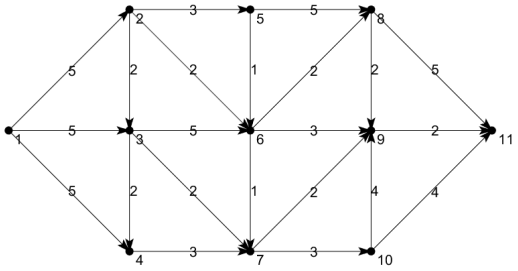

Fig. 2. The digraph with annotated edge weights.

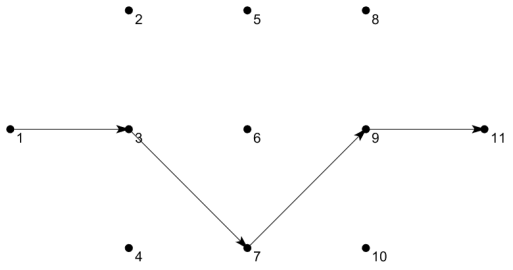

Fig. 3. The minimum edge weight from source to sink.

III. Minimum Vertex Path

A. Formulation

To solve the path for minimum vertex weight, we first find all of the bases and co-bases of the matrix $\mathbf{D}$ containing the vertices's and their weights. Then solve the problem as an Integer Linear Programming (ILP) optimization problem.

B. Example: Minimum Vertex Weight

Using the graph shown in Fig. 1, we can assign vertex weights as shown in Fig. 4 and solve the minimum vertex weight problem using Algorithm 1. The solution is shown in Fig. 5.

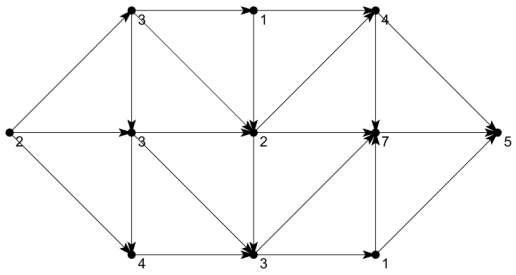

Fig. 4. The digraph with annotated vertex weights.

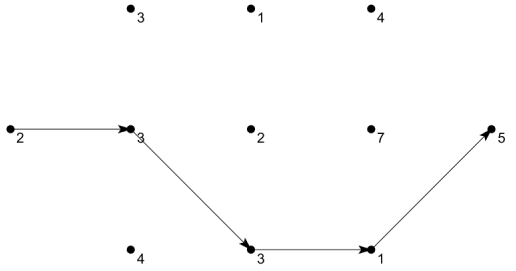

Fig. 5. The minimum vertex weight path from source to sink.

IV. Minimum Vertex Cover

A. Formulation

To solve the minimum vertex cover to fully disconnect a network, we reformulate $\mathbf{D}$, and solve the ILP optimization problem.

B. Example: Minimum Vertex Cover

Using the graph shown in Fig. 1, we can assign vertex weights as shown in Fig. 4 and solve the minimum vertex cover problem to fully disconnect the network. The solution is shown in Fig. 6.

Appendix

A. Directed Graph Definition

- A *self-loop* is an edge that connects a vertex to itself.
- Two edges are *parallel* if they connect the same ordered pair of vertices.
- The *outdegree* of a vertex is the number of edges pointing from it.
- The *indegree* of a vertex is the number of edges pointing to it.
- A *subgraph* is a subset of a digraph's edges (and associated vertices) that constitutes a digraph.

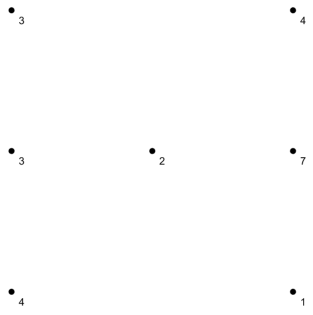

Fig. 6. The minimum vertex cover, i.e. the minimum number of nodes to fully disconnect the network.

- A *directed path* in a digraph is a sequence of vertices in which there is a (directed) edge pointing from each vertex in the sequence to its successor in the sequence, with no repeated edges.
- A directed path is *simple* if it has no repeated vertices.
- A *directed cycle* is a directed path (with at least one edge) whose first and last vertices are the same.
- A directed cycle is *simple* if it has no repeated vertices (other than the requisite repetition of the first and last vertices).
- The *length* of a path or a cycle is its number of edges.
- We say that a vertex w is *reachable from* a vertex v if there exists a directed path from v to w.
- We say that two vertices v and w are *strongly connected* if they are mutually reachable: there is a directed path from v to w and a directed path from w to v.
- A digraph is *strongly connected* if there is a directed path from every vertex to every other vertex.
- A digraph that is not strongly connected consists of a set of *strongly connected components*, which are maximal strongly connected subgraphs.
- A *directed acyclic graph* (or DAG) is a digraph with no directed cycles.

B. Shortest Path Properties

- *Paths are directed*. A shortest path must respect the direction of its edges.
- The weights are not necessarily distances. Geometric intuition can be helpful, but the edge weights weights might represent time or cost.
- *Not all vertices need be reachable*. If t is not reachable from s, there is no path at all, and therefore there is no shortest path from s to t.
- *Negative weights introduce complications*. For the moment, we assume that edge weights are positive (or zero).
- *Shortest paths are normally simple*. Our algorithms ignore zero-weight edges that form cycles, so that the shortest paths they find have no cycles.
- *Shortest paths are not necessarily unique*. There may be multiple paths of the lowest weight from one vertex to another; we are content to find any one of them.
- *Parallel edges and self-loops may be present*. In the text, we assume that parallel edges are not present and use the notation $v \mapsto w$ to refer to the edge from v to w, but our code handles them without difficulty.

REFERENCES

[1] R. Diestel, *Graph Theory*. Springer, Graduate Texts in Mathematics, 2016.
[2] R. Stanley, *Enumerative Combinatorics*. Cambridge University Press, 2000.