I214 System Optimization
Chapter 4: Network Optimization

Brian Kurkoski

Japan Advanced Institute of Science and Technology

2023 January

This is the preliminary version of the slides, will be updated just before the lecture.

Outline

Outline

4.1 Graph Theory Notation

- A *graph* is a pair $G = (V, E)$, where V is a set of *vertices* and E is a set of *edges*.
- If all edges have a direction, then G is called a *directed graph*. If no edge has a direction, then G is called an *undirected graph*. An example of a directed graph and an undirected graph are in Fig. 1.

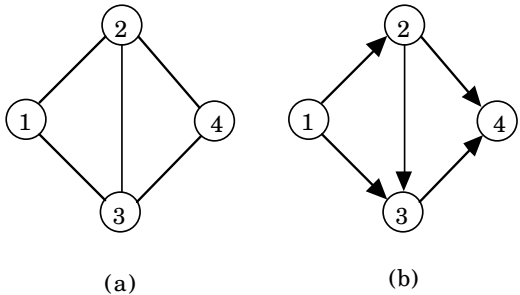


Figure 1: (a) An undirected graph and (b) a directed graph.

Graph Theory Notation

- An edge of a directed graph is an ordered pair (i, j) of vertices, where vertex i is called the *initial vertex* and j is called the *terminal vertex*, for $i \neq j$. An edge of an undirected graph is an unordered pair $\{i, j\}$ for $i \neq j$ where vertices i, j are called *end vertices*.
- A route from some vertex to another vertex following the direction of each edge is called *a walk*. If parallel edges (more than one edges with the same initial and terminal vertices) do not exist, then a walk is specified by a sequence of vertices. In a walk $(v_1, v_2, \cdots, v_n)$, we call v_1 the initial vertex and v_n the terminal vertex.
- If a walk does not go through each edge more than once, it is called *a path*.

Graph Theory Notation

- A *weighted graph* or *network* is a graph where a number (or weight) is assigned to each edge.

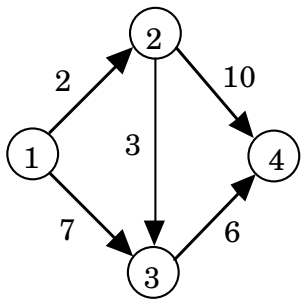


Figure 2: A weighted graph or network.

4.2 Shortest Path Problem

4.2.1 Definition

On a weighted graph or network, the weight of a walk is the sum of the weights on each edge of the walk. Given two vertices on a graph, the *shortest path problem* finds the walk with the minimum weight.

Example

Fig. 3 shows a network of roads that connect seven cities A-G. The number on each edge indicates the distance between two cities. Find a shortest route from A to G.

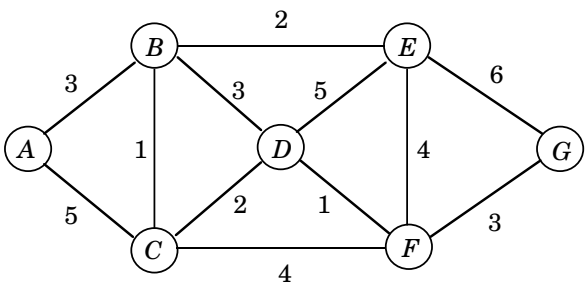


Figure 3: Shortest path problem.

Comments

- Any walk with the smallest length contains no cycles with positive length.
- If a cycle with negative length exists, then we may have a walk with an infinitely small length, i.e., the solution is unbounded. For this reason, we can focus on paths, and not walks.

4.2.2 Dijkstra's Algorithm

Dijkstra's algorithm is a method to find the shortest path in a network. The method is applicable only to networks with non-negative weights.

- Each node i has a label u_i, the label is a number.
- The label represents a distance from the start to that node.
- Initially, the start node has has label $u_1 = 0$. (The distance from the start node to itself is 0).
- For all other nodes, initially, the labels are temporary and initialized with the value ∞.

4.2.2 Dijkstra's Algorithm

Dijkstra's algorithm is a method to find the shortest path in a network. The method is applicable only to networks with non-negative weights.

- Each node i has a label u_i, the label is a number.
- The label represents a distance from the start to that node.
- Initially, the start node has has label $u_1 = 0$. (The distance from the start node to itself is 0).
- For all other nodes, initially, the labels are temporary and initialized with the value ∞.
- As the algorithm progresses, a temporary label value may change.
- When the label changes from temporary to permanent, then its value will no longer change. A permanent label is denoted u_i^*.
- For the network, the weight on the edge (i, j) is d_{ji}.

Dijkstra's Algorithm

1. *Initialize* The start vertex 1 begins with permanent label $u_1^* = 0$. All other label values are $u_j = \infty$, $j \neq 1$.
2. Let i be a vertex with the smallest temporary label, call i the *current vertex*. For all edges (i, j): if vertex j does not have a permanent label and

   $$u_j > u_i^* + d_{ij},$$

   then update the temporary label with

   $$u_j = u_i^* + d_{ij}. \tag{1}$$

3. For the current vertex i, change the temporary label u_i to a permanent label u_i^*.
4. If all nodes have permanent labels, then output all $u_1^*, u_2^*, \ldots$ as the minimal distances. Otherwise, go to Step 2.

Each permanent label u_i^* indicates the length of a shortest path from vertex 1 to vertex i.

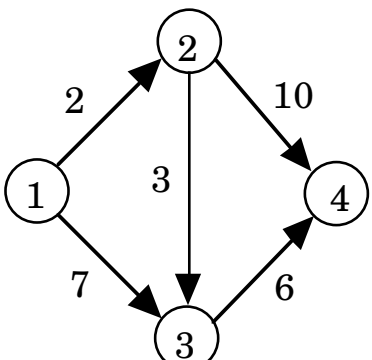
Dijkstra's Algorithm — Find Shortest Path

The shortest path is obtained by adding the following procedure to the algorithm.

- When a temporary label is updated and becomes $u_j = u_i^* + d_{ij}$, the algorithm remembers the link from i to j.
- Denote this link by $\pi(j) = i$. That is, $\pi(j)$ is the neighbor i of j which has the lowest weight.
- Using these π, we can construct a tree from vertex 1 to all other vertices, which indicates a shortest path from vertex 1 to other vertices.

Dijkstra's Algorithm — Example

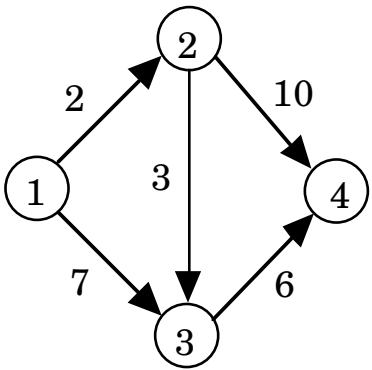
Use Dijkstra's algorithm to find the shortest path from node 1 to node 4.



	u				π			
	1	2	3	4	1	2	3	4
0	0	∞	∞	∞	-			
1								
2								
3								

Dijkstra's Algorithm — Example

Use Dijkstra's algorithm to find the shortest path from node 1 to node 4.



	u				π			
	1	2	3	4	1	2	3	4
0	0	∞	∞	∞	-			
1		$\underline{2}$	7	∞	-	1	1	
2			$\underline{5}$	12	-	1	2	2
3				$\underline{11}$	-	1	2	3

Shortest path has length 11. Shortest path to node 4 is
$4 \leftarrow \pi(4) = 3 \leftarrow \pi(3) = 2 \leftarrow \pi(2) = 1$.

4.2.3 Correctness of Dijkstra's Algorithm

A proof of the correctness of Dijkstra's algorithm is given.

Given that all existing permanent labels are correctly decided, show that for a node i not yet decided, its permanent label u_i^* correctly indicates the length of a shortest path from vertex 1 to vertex i.

4.2.4 Bellman-Ford Algorithm

The Bellman-Ford algorithm also finds the shortest path in a network, but can be applied to networks where some edges have a negative distance.

- Both the Dijkstra algorithm and the Bellman-Ford algorithm maintain a list of labels that indicates the current best distances.
- Dijkstra's algorithm optimizes nodes in serial (i.e. optimizing one at a time) while Bellman-Ford performs all computations in parallel.
- The Bellman-Ford algorithm maintains a list $\mathcal{S}^{(k)}$ of nodes that were updated on iteration k. Since these are the only nodes with new labels, these are used on iteration $k+1$ to reduce the amount of computation needed.

Bellman-Ford Algorithm

1. Label the start vertex 1 $u_1^{(0)} = 0$, and label other vertices $u_j^{(0)} = \infty$, for $j \neq 1$ Let $\mathcal{S}^{(0)} = \{1\}$ and $k = 1$.
2. For each $i \in \mathcal{S}^{(k-1)}$ and each edge (i, j), find the minimum value of

$$u_i^{(k-1)} + d_{ij} \text{ for } i \in S^{k-1},\ (i,j) \in E).$$

   Suppose $u_{i^*}^{(k-1)} + d_{i^*j}$ is the minimum value. If it is less than $u_j^{(k-1)}$, then let $u_j^{(k)} = u_{i^*}^{k-1} + d_{i^*j}$, otherwise let $u_j^{(k)} = u_j^{(k-1)}$. For other j, let $u_j^{(k)} = u_j^{(k-1)}$. The algorithm records the link as $\pi(j) = i^*$.
3. If no labels change, then halt.
4. Let $\mathcal{S}^{(k)}$ be the set of vertices whose labels were updated. Let $k \leftarrow k + 1$ and go to Step 2.

Bellman-Ford Algorithm — Example

Apply the Bellman-Form algorithm to the network in Fig. 5.

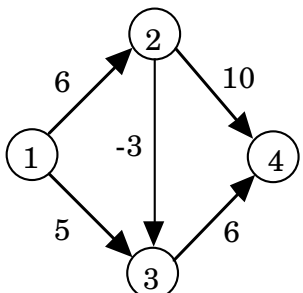


Figure 4: A network with a negative cycle.

	u				π				
k	1	2	3	4	1	2	3	4	S^k
0	0	∞	∞	∞	-				1
1									
2									
3									
4									

Bellman-Ford Algorithm — Example

Apply the Bellman-Form algorithm to the network in Fig. 5.

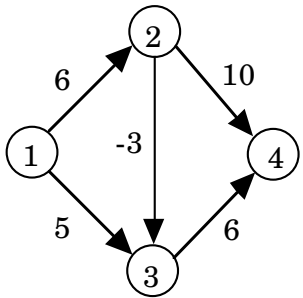


Figure 5: A network with a negative cycle.

	u				π				
k	1	2	3	4	1	2	3	4	S^k
0	0	∞	∞	∞	-				1
1	0	6	5	∞	-	1	1		2, 3
2	0	6	3	11	-	1	2	3	3, 4
3	0	6	3	9	-	1	2	3	4
4	0	6	3	9	-	1	2	3	-

Shortest path has length 9. Shortest path to node 4 is $1 \to 2 \to 3 \to 4$.

Class Info

- Tutorial Hours: Today at 13:30.
- Homework 3 on LMS. Deadline: Friday, January 6 at 18:00.
- Monday, January 9 is Seijin no Hi (Coming of Age Day) — No Class
- Next lecture: Friday, January 9 at 9:00.