Network Optimization: Continuous and Discrete Models

Carlo Fischione

8 February 2014

The Shortest Path Problem (SPP)

We are given a directed graph

$$\mathcal{G} = (\mathcal{N}, \mathcal{A})$$

In SPP, we only consider **Forward Paths**

Consider a forward path

$$(i_1, i_2, \ldots, i_x) = P$$

Length/Cost/Distance of $P = \sum_{x=1}^{k-1} a_{i_x i_{x+1}}$ where $a_{i_x i_{x+1}}$ is the cost of arc $(i_x i_{x+1})$

SPP $\equiv$ what is the forward path having the minimum cost /length /distance?

The Shortest Path Problem (SPP)

$$\min \sum_{(i,j)\in\mathcal{A}} a_{ij}x_{ij}$$

$$\sum_{j|(i,j)\in\mathcal{A}} x_{ij} - \sum_{j|(i,j)\in\mathcal{A}} x_{ji} = \begin{cases} 1 & \text{if } i = s \\ -1 & \text{if } i = t \\ 0 & \text{otherwise} \end{cases}$$

s= source

t= destination

The Shortest Path Problem (SPP)

a_{ij}= delay to t_x from i to j in common network

Any forward path P from s to t can be associated to a flow vector x

$$x_{ij} = \begin{cases} 1 & \text{if } (i,j) \in P \\ 0 & \text{otherwise} \end{cases}$$

Cost of $x = \sum_{i,j \in \mathcal{A}} a_{ij} x_{ij}$

If x is feasible, then the cost of x is the length of P
If the SPP has an optimal solution, then the arc flows x_{ij} are either 0 or 1

The Shortest Path Problem (SPP)

Examples of Application of SPP

1. Routing in Data Network
2. Dynamic Programming
3. Project Management
4. The Paragraphing Problem

A Generic Shortest Path Algorithm

- Consider single origin—many destinations problem
- We would like to find algorithms that solve SPP
- $s \equiv 1$
- $d_i =$ scalar label associated to node i

Proposition 2.1 (optimality condition)

Let $d_1, d_2, \ldots, d_n$ satisfy

$$d_j \leqslant d_i + a_{ij} \quad \forall (i,j) \in \mathcal{A} \quad (2.2)$$

Let P start at node i_1 and end at node i_k, If $d_j = d_i + a_{ij}$ (2.3) $\forall$ arcs (i,j) of P , then P is a shortest path from i_1 to i_k

A Generic Shortest Path Algorithm

Proof

$$\sum_{j \in P} d_j = d_{i_2} + \cdots + d_{i_n} \leqslant d_{i_1} + \cdots + d_{i_{k-1}} + \sum_{i,j \in P} a_{ij} \Rightarrow$$

$$\sum_{j \in P} a_{ij} = d_{i_k} - d_{i_1}$$

However, $\forall\ P' \neq P$ having same start and ending points, from (2.2) we have $\sum_{j \in P'} Q_{ij} \geqslant d_{i_k} - d_{i_1} \Longrightarrow \quad P$ is the shortest path
(2.2) and (2.3) are called *Complementary Slackness Conditions* for the shortest path

A Generic Shortest Path Algorithm

The Complementary Slackness Conditions suggest to set initial price to the labels and then updates it iteratively
$\forall$ iteration d_i is an estimate of the length of some path P_i from 1 to i

Initialization of the Generic Shortest Path Algorithm

$$\text{If } d_j > d_i + a_{ij}, \quad d_j = d_i + aij \begin{cases} \mathcal{V} = \text{candidate list of nodes} \\ \mathcal{V} = 1 \quad \text{initially} \\ d_1 = 0, d_i = \infty \quad \forall i \neq 1 \end{cases}$$

Iterations of the Generic Shortest Path Algorithm

Remove i from $\mathcal{V}$. $\forall (i,j) \in \mathcal{A}$ outgoing from i
if $d_j > d_i + a_{ij}$, $\quad\quad$ $d_j = d_i + a_{ij}$ and add j to $\mathcal{V}$ if not yet in $\mathcal{V}$

- Labels are monotonically non increasing
- $d_i < \infty \iff i$ has entered $\mathcal{V}$ at least once

Example

See Figure 2.3

Iterations of the Generic Shortest Path Algorithm

Proposition 2.2

Consider the Generic Shortest Path algorithm

a) At the end of each iteration, the following conditions hold:
   i If $d_j < \infty$, then d_j is the length of some path that starts at 1 and ends at j
   ii If $i \notin \mathcal{V}$, then either $d_i = \infty$ or $d_j \leqslant d_i + a_{ij} \quad \forall j \in \mathcal{A}$

b) If the algorithm terminates, upon termination,
$\forall j : d_j < \infty, d_j$ is the shortest distance from 1 to j

$$d_j = \begin{cases} \min_{(i,j)\in\mathcal{A}} \quad d_i + a_{ij} & j \neq 1 \\ 0 & j = 1 \end{cases}$$

$d_j = \infty$ if and only if no path from 1 to j

Iterations of the Generic Shortest Path Algorithm

Proposition 2.2 Cont'd

c) If the algorithm does not terminate, there exists some node j and a sequence of paths starting at 1, end at j having lengths that diverge to $-\infty$

d) The algorithm terminates if and only if there is no path that starts at 1 and contains a cycle with negative length.

Proof

a) (i) can be proved by induction on the iteration count.

a) (ii) if $i \notin \mathcal{V}$, d_i stays constant, while d_j has not increased since last iteration has updated $\Rightarrow$
$d_j \leqslant d_i + a_{ij}$, d_j could have been decreased in another iteration

Iterations of the Generic Shortest Path Algorithm

Proof Cont'd

c) If the algorithm dose not terminate, then there exist some d_j that decrease strictly an infinite number of times. Since the number of paths $1 \to j$ is finite, there must be a cycle p_j with negative length. Iterating on such cycle, $d_j \to -\infty$

d) From (c), the algorithm terminates if and only if there is a lower bound on the length of all paths that start at node 1. This is possible only if there is no cycle with negative length

b)

$$\mathcal{I} = \{i \,|d_i < \infty \quad \text{upon termination}\}$$

$$\bar{\mathcal{I}} = \{i \,|d_i = \infty \quad \text{upon termination}\}$$

Iterations of the Generic Shortest Path Algorithm

Proof Cont'd

b) If $j \in \bar{\mathcal{I}}$, there can not be path $i \to j$
If $j \in \mathcal{I}$ then (2.4) implies $d_j \leqslant d_i + a_{ij} \quad \forall j : (i,j) \in \mathcal{A}$
$\Rightarrow$ there exists P_j from 1 to j:

$$\sum_{j \in P_j} d_j \leqslant \sum_{j \in P_j} d_n + d_1 - d_j + \sum_{i,j \in P_j} a_{ij} \Rightarrow$$

$d_j - d_1 \leqslant \sum_{i,j \in P_j} a_{ij} \Rightarrow P_j$ is the shortest path

- Consequence
  - If there are cycles having arcs with negative lengths, the algorithm may diverge
  - If there are cycles with arc lengths positive, the algorithm will converge

Proof Cont'd

This is used to build routes in routing protocols for common networks

- When all cycles have nonnegative length and there exists a path from 1 to every node j, the algorithm terminates and gives

$$d_j = \min_{(i,j)\in\mathcal{A}}\{d_i + a_{ij}\} \quad \forall j \neq 1 \quad \textit{Bellman's Equation}$$

  It suggests how to build the paths so they say who is the shortest path

- The Generic Shortest Path Algorithm gives freedom to select $\mathcal{V}$ at the beginning and how to remove nodes from $\mathcal{V}$. The node removal rules can be grouped into two classes

a) Label setting (E.g Dijkstra Methods)

b) Label correcting (E.g Bellman Ford Methods)

Iterations of the Generic Shortest Path Algorithm

Label Setting (Dijkstra) Methods

$$d_i = \min_{j \in \mathcal{V}} d_j \qquad i \text{ is removed}$$

$$\mathcal{V} = \{1\}$$

$$d_1 = 0, \quad d_i = \infty \quad \forall i \neq 1$$

Iteration of the Label Setting Method

Remove from $\mathcal{V}$ $\quad i : d_i = min_{j \in \mathcal{V}} d_j$
$\forall$ outgoing arc $(i,j) \in \mathcal{A}$, if $d_j > d_i + a_{ij}*$ set $d_j = d_i + a_{ij}$ and add j to $\mathcal{V}$ if it does not belong to $\mathcal{V}$

Example

See Figure 2.5

Iterations of the Generic Shortest Path Algorithm

Proposition 2.3

Assume that all arc lengths are nonnegative

a) For any iteration of the Label Setting Method, we have

$$\mathcal{W} = \{i | d_i < \infty, i \notin \mathcal{V}\}$$

(i) No node of $\mathcal{W}$ at the start of the iteration will enter the candidate list $\mathcal{V}$ during the iteration.

(ii) At the end of the iteration, $d_i \leqslant d_j \ \forall i \in \mathcal{W}$ and $j \notin \mathcal{W}$

(iii) For each node j, consider the simple path starting at 1 and ending at j are having all their nodes in $\mathcal{W}$ at the end of the iteration. Then d_j at the end of the iteration is equal to the length of the shortest of these paths ($d_j = \infty$ if no such path exists)

Iterations of the Generic Shortest Path Algorithm

Proposition 2.3 Cont'd

b) The label setting will terminate, and all nodes with a final label that is finite will be removed from the candidate list $\mathcal{V}$ exactly once in order of increasing shortest distance from node 1; that is, is the final labels of i and j are finite and satisfy $d_i < d_j$, then i will be removed before j

Proof

(i) and (ii) are proved together by induction. Clearly (i) and (ii) hold for the initial iteration at which node 1 exits $\mathcal{V}$ and enters $\mathcal{W}$

(i) By induction, consider node i at the beginning of an iteration. Then

$$d_i^{(k+1)} = \min_{j \in \mathcal{V}} d_j^{(k)} \begin{cases} d_i^{(k+1)} = d_i \text{ at beginning of iteration} \\ d_i^{(k)} = \text{value of } d_j \end{cases}$$

Iterations of the Generic Shortest Path Algorithm

Proof proposition 2.3 Cont'd

(i) By induction, we assume $d_j^{(k)} \leqslant d_i^{(k+1)} \forall j \in \mathcal{W}$ and $a_j \geqslant 0$ at the beginning of the irritation
Can $j \in \mathcal{W}$ enter $\mathcal{V}$? No, because it should be $d_j^{(k)} > d_i^{(k+1)} + a_{ij}$, for $*$, which is not possible because $d_j^{(k)} \leqslant d_i^{(k+1)}$

(ii) At the end of iteration k , can we have $d_i^{(k+1)} > d_j^{(k+1)}$ $i \in \mathcal{W}, j \notin \mathcal{W}$? No, because for $*$ $d_j^{(k+1)} = \min(d_j^{(k)}, d_i^{(k+1)} + a_{ij})$ $\forall j \in$ outgoing of i

Iterations of the Generic Shortest Path Algorithm

Proof proposition 2.3 Cont'd

(ii) $d_{i\in\mathcal{W}}^{(k+1)} = \min_{j\in\mathcal{V}}(d_j^{(k)}) \leqslant \min_{\forall J\ outgoing}(d_j^{(k)})$ because either $j \in \mathcal{V}$ or $j \in \mathcal{W}$ (labels are non increasing) or $j \in \bar{\mathcal{W}}$ (labels $=\infty$)

$$d_{j\in\text{outgoing } i}^{(k+1)} \geqslant (d_i^{(k+1)}, d_i^{(k+1)} + a_{ij})$$

$$\geqslant d_{i\in\mathcal{W}}^{(k+1)}$$

(iii) If we apply Proposition 2.2 to the subgraph consisting of the nodes of $\mathcal{W} \cup j$ and arcs that have both end nodes in $\mathcal{W} \cup j$, the rest follows immediately

Iterations of the Generic Shortest Path Algorithm

Proof proposition 2.3 Cont'd

(b) Once a node enters $\mathcal{W}$, stays there and never returns $\mathcal{V}$ again. Every time we pick the minimum i and $\mathcal{W}$ consists the labels of those nodes that was already removed from $\mathcal{V}$ because of lower label (property (ii)), it follows that the end with which nodes enter $\mathcal{W}$ has incoming labels

Label Correcting Methods –The Bellman Ford Method

The rules to nodes from $\mathcal{V}$ is simple, but a node may return multiple times in $\mathcal{V}$

$\mathcal{V}=$ queue of nodes $\begin{cases} \text{a node is always removed from the top} \\ \text{a node is always entered at the bottom} \end{cases}$

Iterations of the Generic Shortest Path Algorithm

Bellman-Ford Property

For each node i and integer $k \geqslant 1$, let d_i^k = shortest distance from 1 to i using paths that have k arcs or less
$d_i^k = \infty$ if there is no path 1 or i with k arcs or less. Then d_i at the end of the $k-$th cycle of irritations of Bellman-Ford method is less or equal to d_i^k

This property is very useful because the Bellman-Ford algorithm converges when all cycles have nonnegative length(Since the algorithm is just a particular case of the Generic Shortest Path Algorithm that Proposition 2.2 holds)
Since there are (simple) paths with at most $N-1$ arcs in a $\mathcal{N}$ nodes graph, the Bellman-Ford property implies that the method finds all the shortest distances in at most $N-1$ cycles

Iterations of the Generic Shortest Path Algorithm

Bellman-Ford Property

The method can be used to detect the presence of a negative cycle: if the method does not terminate at the end of $N-1$ iterations, there are cycles.

Proof of the property

Simple, by iteration and using proposition 2.2 (a)

Comparison of Label Setting and Label Correcting

- It can be proved that label setting methods have worst-case running time better than label correcting methods

Iterations of the Generic Shortest Path Algorithm

Comparison of Label Setting and Label Correcting

- It can be proved that label setting methods have worst-case running time better than label correcting methods
- However, for cyclic graphs, label correcting methods can be improved so that nodes only enter in $\mathcal{V}$ once and thus have better performance than label setting methods (this is the case of dynamic programming where there is an acyclic graph)
- When the graph is sparse, label correcting methods have practical running time than label setting methods.