the Shortest Path Problem (SPP)

$G = (N, A)$

In SPP we only consider forward paths

Consider a forward path

$(i_1, i_2, \dots, i_k) = P$

Length of P $= \sum_{n=1}^{k-1} a_{i_n i_{n+1}}$ — Sum of the costs of the arcs of P

or

Cost of P

or

Distance of P

$a_{i_n i_{n+1}}$ → cost of arc (i_n, i_{n+1})

SSP = what is the forward path having the minimum cost / length / distance?

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

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

s = source

t = destination

Example

a_{ij} = delay to tx from i to j in comm. network

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

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

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

If x is feasible, then the cost of x is the length of P

Result

If the SPP has an opt. sol, then the arc flows x_{ij} are either 0 or 1.

Examples of Application of SPP

1) Routing in Data Network
2) Dynamic Programming
3) Project Management
4) Pentagraphing Problem

A Generic Shortest Path Algorithm

- Consider single origin - many destination problems
- We would like to find algorithms that solve SSP
- $s \triangleq 1$
- d_i = scalar label associated to node i

Proposition 2.1 (optimality condition)

Let $d_1, d_2, \dots d_N$ satisfy

$$\boxed{d_j \leq d_i + a_{ij}} \quad \forall (i,j) \in A \qquad (2.2)$$

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

Proof

(2.3)

$$\sum_{j \in P} d_j = d_{i_2} + \dots + d_{i_K} = d_{i_1} + \dots + d_{i_{K-1}} + \sum_{j \in P} a_{ij} \Rightarrow$$

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

However $\forall P' \neq P$ having same start and ending points, from (2.2) we have

$$\sum_{j \in P'} a_{ij} \geq d_{i_K} - d_{i_1} \Rightarrow P \text{ is the shortest path}$$

(2.2) and (2.3) are called complementary slackness conditions for the shortest path

- the Complementary Slackness Condition suggest to set initial values to the labels and then update iteratively.
- $\forall$ iteration d_i is an estimate of the length of some path P_i from 1 to i
- If $d_j > d_i + a_{ij}$, $d_j := d_i + a_{ij}$

V = candidate list of nodes
$V = \{1\}$ initially
$d_1 = 0$, $d_i = \infty \ \forall \ i \neq 1$

} Initialization of the Generic Shortest Path Algorithm

Iterations of the Generic Shortest Path Algorithm

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

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

Example

See Figure 2.3

Proposition 2.2 Consider the generic shortest path Algo

a) at the end of every iteration, we it holds

(i) if $d_j < \infty$, d_j is the lenght of some path from 1 to j

(ii) if $i \notin V$, then either $d_i = \infty$ or

$$d_j \leq d_i + a_{ij} \quad \forall j \in A$$

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

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

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

c) If the alg does not terminate, then $\exists$ some node j and a sequence of paths starting at 1 and ending at j having length diverging to $-\infty$

d) the algorithm terminates if and only if there is no path starting at 1 containing a cycle with negative lenght

---

Proof

a) (i), trivial by induction

a) (ii) if $i \notin V$, d_i stays constant, whereas d_j has not increased since last time it was updated $\Rightarrow$ $d_j \leq d_i + a_{ij}$

(c) if the algo does not terminate, then there $\exists$ some d_j that decreases strictly an infinite amount of times. Since the number of paths $1 \to j$ is finite, there must be a cycle P_j with negative lenght. Iterating on such cycle, $d_j \to -\infty$

(d) From (c), the algo terminates if and only if there is a lower bound on the lenght of all paths from 1. this is possible only if there is no cycle with negative lenght

(3) $I = \{ i \mid d_i < \infty \text{ upon termination} \}$

$\bar{I} = \{ i \mid d_i = \infty \text{ upon termination} \}$

- if $j \in \bar{I}$, then ~~$\exists$~~ cannot be path $i \to j$
- if $j \in I$ then (2.4) implies

$d_j \leq d_i + a_{ij} \quad \forall j : (i,j) \in A$

$\Rightarrow \exists P_j$ from 1 to j:

$\sum_{k \in P_j} d_k \leq \sum_{k \in P_j} d_k + d_1 - d_j + \sum_{i,k \in P_j} a_{ik} \Rightarrow$

$d_j - d_1 \leq \sum_{i,k \in P_j} a_{ik} \Rightarrow$ P_j is the shortest path.

Label Setting (Dijkstra) Methods

$d_i = \min_{j \in V} d_j$ $\quad$ i node is removed

$V = \{1\}$

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

Iteration of the Label Setting Method

remove from V i : $d_i = \min_{j \in V} d_j$

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

Example

See Fig 2.5

Proposition 2.3 Assume that all arc lengths are non negative

a) For any iteration of the label setting method, we have

$W = \{ i \mid d_i < \infty, i \notin V \}$ set of permanently labeled nodes

(i) No node of W at the start of the iteration will enter the candidate list V during the iteration. (Thus W is the set of permanently labelled nodes)

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

(iii) For each node j, consider the simple path starting at 1 and ending at j an having all their nodes in 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)

(b) The label setting will terminate, and all nodes with a final label that is finite will be removed from the candidate list V exactly once in order of increasing shortest distance from node 1. That is, if 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 by induction. Clearly (i) and (ii) hold for the initial iteration at which node 1 exits V and enters W.

(i) By induction, consider node i at the beginning of an iteration k. Then $\begin{cases} d_i^{(k+1)} = \text{value of } d_i \text{ at beginning of iteration } k \\ d_j^{(k)} = \text{value of } d_j \neq i \text{ at iteration } k \end{cases}$

$d_i^{(k+1)} = \min_{j \in V} d_j^{(k)}$

By induction, we have $d_j^{(k)} \geq d_i^{(k+1)} \; \forall j \in W$ and $a_{ij} \geq 0$ at the beginning of the iteration.

Can $j \in W$ enter V? No, because it should be $d_j^{(k)} > d_i^{(k+1)} + a_{ij}$ for (*), which is not possible because $d_j^{(k)} \geq d_i^{(k+1)}$.

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

(*) $d_j^{(k+1)} = \min\left(d_j^{(k)}, d_i^{(k+1)} + a_{ij}\right)$ $j \notin W$ when

$d_i^{(k+1)} = \min\left(d_j^{(k)}\right)$ $j \in V$ $V \subset \bar{W} \Rightarrow$

$d_j^{(k+1)} \leq d_i^{(k+1)}$ because

$$d_j^{(k+1)} = \min\left(d_j^{(k)}, d_i^{(k+1)} + a_{ij}\right) =$$

$$= \min\left(d_j^{(k)}, \min_{j \in V}\left(d_j^{(k)}\right) + a_{ij}\right)$$

$$\leq \min\left(d_j^{(k)}, d_j^{(k)} + a_{ij}\right) \leq d_i^{(k+1)}$$
$j \notin W$ $\quad$ $j \in V$ $\quad$ $\bar{W} \supset V$

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

(b) Once a node enters W, stays there and never enters V again. Since every time we pick the minimum i and W contains the labels of those nodes that were already removed from V because of lower label (property (ii)), it follows that the order with with nodes exit W has increasing labels. Prop 2.2 (d) ensures the termination.

Label Correcting Methods

the Bellman-Ford Method

- the rule to nodes from V is simple, but a node may enter multiple times in V.
- V = queue of nodes { a node is always removed from the top; a node is always added at the bottom }

Bellman-Ford property

For each node i and integer $k \geq 1$, 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 to i with k arcs or less.

Then, d_i at the end of the k-th cycle of iterations 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 non negative length (since the algorithm is just a particular case of the Generic Shortest Path Algorithm and Prop 7.2 holds)
- Since there are (simple) paths with at most $N-1$ arcs in a N nodes graph, the Bellman-Ford property implies that the method finds all the shortest distances in at most $N-1$ cycles
- the method can be used to detect the presence of a negative cycle: if the method does not terminate after $N-1$ iterations, there are cycles

Proof of the property

Simple, by induction and using Proposition 7.2(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.
- However, for acyclic graphs, label correcting methods can be improved so that nodes enter only once in V and thus have no worse performance than label setting methods. [This is the case of dynamic programming, where there are no cycles]
- When the graph is sparse, label correcting methods have practical running time better than label setting.