# Chapter-2: Routing Algorithms: Shortest Path and Widest Path

Deep Medhi and Karthik Ramasamy

August 2007

http://www.NetworkRouting.net





# Introduction

- Routing algorithms for different purpose
  - Shortest Path
    - Path that has the least cost
  - Widest Path
    - Path that has the most width
- For each purpose, different approaches available
  - To cover a few key algorithms used in communication network routing

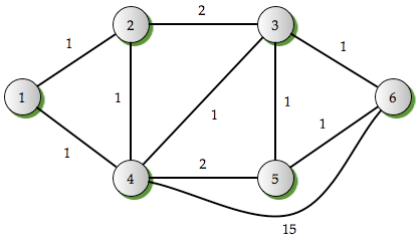
# Notation Summary (more at the end)

| Algorithm | Indicator | Additive | Nonadditive (Widest) |
|---|---|---|---|
| Bellman–Ford | "overbar" | $\overline{D}_{ij}$ | $\overline{B}_{ij}$ |
| Dijkstra | "underscore" | $\underline{D}_{ij}$ | $\underline{B}_{ij}$ |
| Path caching | "hat" | $\hat{D}_{ij}$ | $\hat{B}_{ij}$ |

3

# A simple illustration

- Cost of each link given




- By eyeballing we can see that the shortest path from node-1 to node-6 is 1-4-3-6 with a cost of 3 units
  - How to find this from an algorithmic standpoint?

4

# Bellman-Ford Approach

- Centralized view
- Track two sets of information

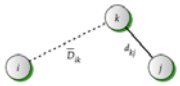
$d_{ij}$ = Link cost between nodes $i$ and $j$

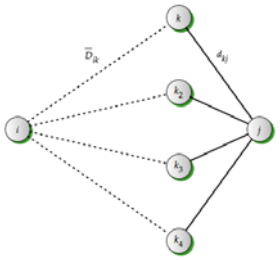
$\overline{D}_{ij}$ = Cost of the computed minimum cost path from node $i$ to node $j$.

- Bellman-Ford equation

$$\overline{D}_{ii} = 0, \quad \text{for all } i,$$

$$\overline{D}_{ij} = \min_{k \neq j} \{\overline{D}_{ik} + d_{kj}\}, \quad \text{for } i \neq j.$$




- Not an algorithm
  - Know somehow $\overline{D}_{ik}$

5

# Bellman-Ford Algorithm

- Centralized Approach: use an iterative scheme using the number of hops; consider $\overline{D}_{ij}^{(h)}$ = cost of the minimum cost path from node $i$ to node $j$ when up to $h$ number of hops are considered.

ALGORITHM 2.1 **Bellman–Ford centralized algorithm.**

Initialize for nodes $i$ and $j$ in the network:

$$\overline{D}_{ii}^{(0)} = 0, \quad \text{for all } i; \qquad \overline{D}_{ij}^{(0)} = \infty, \quad \text{for } i \neq j. \tag{2.2.2a}$$

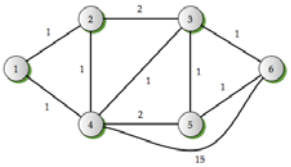
For $h = 0$ to $N - 1$ do

$$\overline{D}_{ii}^{(h+1)} = 0, \quad \text{for all } i \tag{2.2.2b}$$

$$\overline{D}_{ij}^{(h+1)} = \min_{k \neq j} \left\{\overline{D}_{ik}^{(h)} + d_{kj}\right\}, \quad \text{for } i \neq j. \tag{2.2.2c}$$

6

# Illustration




- From 1 to 6:
  - $\overline{D}_{16}^{(1)} = \infty$
  - $\overline{D}_{16}^{(2)} = \overline{D}_{14}^{(1)} + d_{46}$

$k = 3$: $\overline{D}_{13}^{(2)} + d_{36} = 2 + 1 = 3$ ←

$k = 5$: $\overline{D}_{15}^{(2)} + d_{56} = 3 + 1 = 4$

$k = 4$: $\overline{D}_{14}^{(2)} + d_{46} = 1 + 15 = 16.$

**TABLE 2.1** Minimum cost from node 1 to other nodes using Algorithm 2.1.

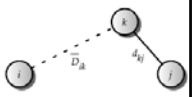
| $h$ | $\overline{D}_{12}^{(h)}$ | Path | $\overline{D}_{13}^{(h)}$ | Path | $\overline{D}_{14}^{(h)}$ | Path | $\overline{D}_{15}^{(h)}$ | Path | $\overline{D}_{16}^{(h)}$ | Path |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | $\infty$ | – | $\infty$ | – | $\infty$ | – | $\infty$ | – | $\infty$ | – |
| 1 | 1 | 1-2 | $\infty$ | – | 1 | 1-4 | $\infty$ | – | $\infty$ | – |
| 2 | 1 | 1-2 | 2 | 1-4-3 | 1 | 1-4 | 3 | 1-4-5 | 16 | 1-4-6 |
| 3 | 1 | 1-2 | 2 | 1-4-3 | 1 | 1-4 | 3 | 1-4-5 | 3 | 1-4-3-6 |
| 4 | 1 | 1-2 | 2 | 1-4-3 | 1 | 1-4 | 3 | 1-4-5 | 3 | 1-4-3-6 |
| 5 | 1 | 1-2 | 2 | 1-4-3 | 1 | 1-4 | 3 | 1-4-5 | 3 | 1-4-3-6 |

7

# Distance Vector Approach

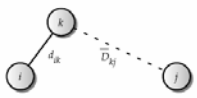
- Presents a complementary view from Bellman-Ford
  - Bellman-Ford: I'll find to k, get met k-j link distance

$$\overline{D}_{ij} = \min_{k \neq j} \{\overline{D}_{ik} + d_{kj}\}, \quad \text{for } i \neq j.$$




  - Distance Vector: I've direct link to k; k finds distance k to i (somehow)

$$\overline{D}_{ij} = \min_{k \neq i} \{d_{ik} + \overline{D}_{kj}\}, \quad \text{for } i \neq j.$$




  - Note the change in order of information; allows to take a completely distributed view

8

# Distance Vector Algorithm

– At time $t$, node $i$ can only know what it has received from $k$ regarding distance to node $j$, i.e., $D^i_{kj}(t)$

ALGORITHM 2.2 **Distance vector algorithm (computed at node $i$).**

Initialize

$$\overline{D}_{ii}(t) = 0; \quad \overline{D}_{ij}(t) = \infty, \quad \text{(for node } j \text{ that node } i \text{ is aware of).} \tag{2.2.4a}$$

For (nodes $j$ that node $i$ is aware of) do

$$\overline{D}_{ij}(t) = \min_{k \text{ directly connected to } i} \left\{ d_{ik}(t) + \overline{D}^i_{kj}(t) \right\}, \quad \text{for } j \neq i. \tag{2.2.4b}$$


9

# Illustration

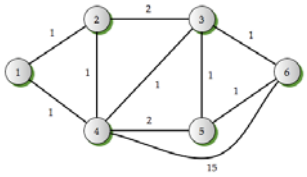



**TABLE 2.2** Distance vector based distributed computation at time $t$ from node 1 to node 6.

| Time, $t$ | $\overline{D}^1_{46}(t)$ | $\overline{D}^1_{26}(t)$ | Computation at node 1 $\min\{d_{14}(t) + \overline{D}^1_{46}(t), d_{12}(t) + \overline{D}^1_{26}(t)\}$ | $\overline{D}_{16}(t)$ |
|---|---|---|---|---|
| 0 | $\infty$ | $\infty$ | $\min\{1+\infty, 1+\infty\}$ | $\infty$ |
| 1 | 15 | $\infty$ | $\min\{1+15, 1+\infty\}$ | 16 |
| 2 | 2 | 3 | $\min\{1+2, 1+3\}$ | 3 |

10

# Dijstra's algorithm

- Computing path from a source node view
  - Need to have the link costs of all links

$d_{ij}$ = link cost between node $i$ and node $j$

$\underline{D}_{ij}$ = cost of the minimum cost path between node $i$ and node $j$.

11

# Dijkstra's algorithm

ALGORITHM 2.3 **Dijkstra's shortest path first algorithm (centralized approach).**

1. Start with source node $i$ in the permanent list of nodes considered, i.e., $\mathcal{S} = \{i\}$; all the rest of the nodes are put in the tentative list labeled as $\mathcal{S}'$. Initialize

   $$\underline{D}_{ij} = d_{ij}, \quad \text{for all } j \in \mathcal{S}'.$$

2. Identify a neighboring node (intermediary) $k$ not in the current list $\mathcal{S}$ with the minimum cost path from node $i$, i.e., find $k \in \mathcal{S}'$ such that $\underline{D}_{ik} = \min_{m \in \mathcal{S}'} \underline{D}_{im}$.

   Add $k$ to the permanent list $\mathcal{S}$, i.e., $\mathcal{S} = \mathcal{S} \cup \{k\}$,

   Drop $k$ from the tentative list $\mathcal{S}'$, i.e., $\mathcal{S}' = \mathcal{S}' \backslash \{k\}$.

   If $\mathcal{S}'$ is empty, stop.

3. Consider the list of neighboring nodes, $\mathcal{N}_k$, of the intermediary $k$ (but do not consider nodes already in $\mathcal{S}$) to check for improvement in the minimum cost path, i.e., for $j \in \mathcal{N}_k \cap \mathcal{S}'$

   $$\underline{D}_{ij} = \min\{\underline{D}_{ij}, \underline{D}_{ik} + d_{kj}\}. \qquad (2.3.1)$$

   Go to Step 2.

12

# Iterative Steps (Dijkstra's Algorithm)

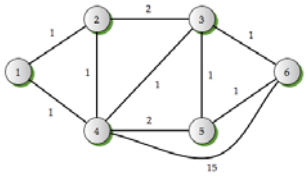
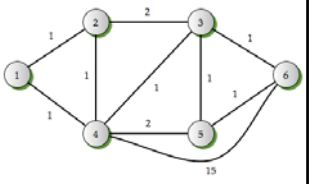


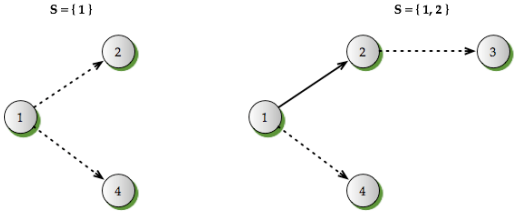

**TABLE 2.3** Iterative steps in Dijkstra's algorithm.

| Iteration | List, $S$ | $D_{12}$ | Path | $D_{13}$ | Path | $D_{14}$ | Path | $D_{15}$ | Path | $D_{16}$ | Path |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | {1} | 1 | 1-2 | ∞ | – | 1 | 1-4 | ∞ | – | ∞ | – |
| 2 | {1, 2} | 1 | 1-2 | 3 | 1-2-3 | 1 | 1-4 | ∞ | – | ∞ | – |
| 3 | {1, 2, 4} | 1 | 1-2 | 2 | 1-4-3 | 1 | 1-4 | 3 | 1-4-5 | 16 | 1-4-6 |
| 4 | {1, 2, 4, 3} | 1 | 1-2 | 2 | 1-4-3 | 1 | 1-4 | 3 | 1-4-5 | 3 | 1-4-3-6 |
| 5 | {1, 2, 4, 3, 5} | 1 | 1-2 | 2 | 1-4-3 | 1 | 1-4 | 3 | 1-4-5 | 3 | 1-4-3-6 |
| 6 | {1, 2, 4, 3, 5, 6} | 1 | 1-2 | 2 | 1-4-3 | 1 | 1-4 | 3 | 1-4-5 | 3 | 1-4-3-6 |

13

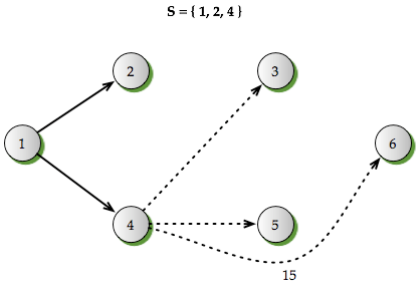
# Illustration

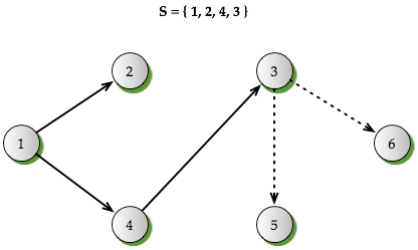









14

# Dijkstra's Algorithm: Distributed approach

- Actually similar to centralized algorithm:
  - The computation of the shortest path by each node is done separately based on link cost information received and available at a node
- Link cost seen by node *i* at time *t* regarding link *l-m:*

  $d^i_{km}(t)$

15

ALGORITHM 2.4 **Dijkstra's shortest path first algorithm (a distributed approach).**

1. Discover nodes in the network, $\mathcal{N}$, and cost of link $k$-$m$, $d^i_{km}(t)$, as known to node $i$ at the time of computation, $t$.
2. Start with source node $i$ in the permanent list of nodes considered, i.e., $\mathcal{S} = \{i\}$; all the rest of the nodes are put in the tentative list labeled as $\mathcal{S}'$. Initialize

   $$\underline{D}_{ij}(t) = d^i_{ij}(t), \quad \text{for all } j \in \mathcal{S}'.$$

3. Identify a neighboring node (intermediary) $k$ not in the current list $\mathcal{S}$ with the minimum cost path from node $i$, i.e., find $k \in \mathcal{S}'$ such that $\underline{D}_{ik}(t) = \min_{m \in \mathcal{S}'} \underline{D}_{im}(t)$.

   Add $k$ to the permanent list $\mathcal{S}$, i.e., $\mathcal{S} = \mathcal{S} \cup \{k\}$,

   Drop $k$ from the tentative list $\mathcal{S}'$, i.e., $\mathcal{S}' = \mathcal{S}' \backslash \{k\}$.

   If $\mathcal{S}'$ is empty, stop.
4. Consider neighboring nodes $\mathcal{N}_k$ of the intermediary $k$ (but do not consider nodes already in $\mathcal{S}$) to check for improvement in the minimum cost path, i.e., for $j \in \mathcal{N}_k \cap \mathcal{S}'$

   $$\underline{D}_{ij}(t) = \min\{\underline{D}_{ij}(t), \underline{D}_{ik}(t) + d^i_{kj}(t)\}. \qquad (2.3.2)$$

   Go to Step 3.

16

# Discussion

- Algorithm 2.4 is a descriptive view of Dijkstra' algorithm
- From an implementation point of view, Dijkstra's algorithm can be stated another way, especially to track the next-hop, an important requirement in IP networks
  - Next hop from node $i$ to node $j$ : $H_{ij}$

17

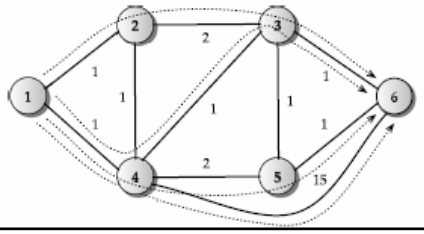
**ALGORITHM 2.5 Dijkstra's shortest path first algorithm (with tracking of next hop).**

```
0   // Computation at time t
1   S = {i}    // permanent list; start with source node i
2   S' = N \ {i}    // tentative list (of the rest of the nodes)
3   for (j in S') do
4     if (d^i_ij(t) < ∞) then    // if i is directly connected to j
5       D_ij(t) = d^i_ij(t)
6       H_ij = j    // set i's next hop to be j
7     else
8       D_ij(t) = ∞
9       H_ij = −1       // next hop not set
10    endif
11  endfor
12  while (S' is not empty) do    // while tentative list is not empty
13    Dtemp = ∞    // find minimum cost neighbor k
14    for (m in S') do
15      if (D_im(t) < Dtemp) then
16        Dtemp = D_im(t)
17        k = m
18      endif
19    endfor
20    S = S ∪ {k}    // add to permanent list
21    S' = S' \ {k}    // delete from tentative list
22    for (j in N_k ∩ S') do
23      if (D_kj(t) > D_ik(t) + d^i_kj(t)) then    // if cost improvement via k
24        D_ij(t) = D_ik(t) + d^i_kj(t)
25        H_ij = H_ik    // next hop for destination j; inherit from k
26      endif
27    endfor
28  endwhile
```

18

# Shortest Path with Path caching

- In certain situations, possible or candidate paths to a destination are known ahead of time
  - In this case, only the updated link cost are needed
  - Then, the computation is fairly simple
- Such situations arise if a router cashes a path, or more commonly, in off-line traffic engineering computation
- Consider the four paths from 1 to 6:




| Path | Cost |
|---|---|
| 1-2-3-6 | $d_{12} + d_{23} + d_{36} = 4$ |
| 1-4-3-6 | $d_{14} + d_{43} + d_{36} = 3$ |
| 1-4-5-6 | $d_{14} + d_{45} + d_{56} = 4$ |
| 1-4-6 | $d_{14} + d_{46} = 16$ |

19

**ALGORITHM 2.6** **Shortest path computation when candidate paths are known.**

*At source node i, a list of candidate paths $\mathcal{P}_{ij}$ to destination node j is available, and link cost, $d^i_{lm}(t)$, of link l-m at time t is known:*

// Initialize the least cost:

$\widehat{D}_{ij}(t) = \infty$

// Consider each candidate path in the list

for ($p$ in $\mathcal{P}_{ij}$) do

  $\widehat{D}_{ij/p}(t) = 0$

  for (link $l$-$m$ in path $p$) do  // add up cost of links for this path

    $\widehat{D}_{ij/p}(t) = \widehat{D}_{ij/p}(t) + d^i_{lm}(t)$  (2.5.2)

  end for

  if ($\widehat{D}_{ij/p}(t) < \widehat{D}_{ij}(t)$) then  // if this is cheaper, note it

    $\widehat{D}_{ij}(t) = \widehat{D}_{ij/p}(t)$

    $\widehat{p} = p$

  end if

end do

20

# Widest path routing

- Widest path routing has been around since the early days of dynamic call routing in the telephone network
- The notion here is different from shortest path routing in that the path with the maximum width is to be chosen
  - Since a path is made of links, the link with the smallest width dictates the width of the path
  - Consider width (bandwidth) of link *l-m* as seen by node *i* at time *t*: $b^i_{km}(t)$

$$\widehat{B}_{ij/p}(t) = \min_{\text{link } l\text{-}m \text{ in path } p} \left\{ b^i_{lm}(t) \right\}$$

  - Main implication: path cost is non-additive (concave)
- Easy to illustrate with path caching example

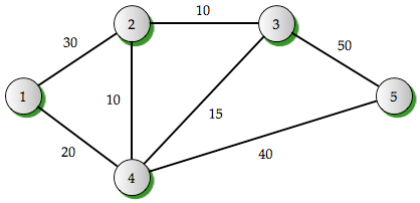


ALGORITHM 2.7 **Widest path computation (non-additive, concave) when candidate paths are known.**

*At source node i, a list of candidate paths $\mathcal{P}_{ij}$ to destination node j is available, and link bandwidth, $b^i_{lm}(t)$, of link l-m at time t is known:*
// Initialize the least bandwidth:
$\widehat{B}_{ij}(t) = 0$
for $p$ in $\mathcal{P}_{ij}$ do
  $\widehat{B}_{ij/p}(t) = \infty$
  for (link *l-m* in path $p$) do  // find bandwidth of the bottleneck link
    $\widehat{B}_{ij/p}(t) = \min\left\{\widehat{B}_{ij/p}(t), b^i_{lm}(t)\right\}$  (2.6.2)
  end for
  if ($\widehat{B}_{ij/p}(t) > \widehat{B}_{ij}(t)$) then  // if this has more bandwidth, note it
    $\widehat{B}_{ij}(t) = \widehat{B}_{ij/p}(t)$
    $\widehat{p} = p$
  end if
end do

# Illustration

- From 1 to 5:




| Path | Cost |
|---|---|
| 1-2-3-5 | $\min\{b_{12}, b_{23}, b_{35}\} = 10$ |
| 1-4-3-5 | $\min\{b_{14}, b_{43}, b_{35}\} = 15$ |
| 1-4-5 | $\min\{b_{14}, b_{45}\} = 20$ |

Widest path is 1-4-5

23

- Can we modify Dijkstra's algorithm and the Bellman-Ford algorithm for widest path routing?
  - Yes, but need a few changes for address non-additive concave cost

24

**ALGORITHM 2.8** **Widest path algorithm, computed at node $i$ (Dijkstra-based).**

1. Discover list of nodes in the network, $\mathcal{N}$ and available bandwidth of link $k$-$m$, $b^i_{km}(t)$, as known to node $i$ at the time of computation, $t$.
2. Initially, consider only source node $i$ in the set of nodes considered, i.e., $\mathcal{S} = \{i\}$; mark the set with all the rest of the nodes as $\mathcal{S}'$. Initialize

   $$\underline{B}_{ij}(t) = b^i_{ij}(t).$$

3. Identify a neighboring node (intermediary) $k$ not in the current list $\mathcal{S}$ with the maximum bandwidth from node $i$, i.e., find $k \in \mathcal{S}'$ such that $\underline{B}_{ik}(t) = \max_{m \in \mathcal{S}'} \underline{B}_{im}(t)$

   Add $k$ to the list $\mathcal{S}$, i.e., $\mathcal{S} = \mathcal{S} \cup \{k\}$

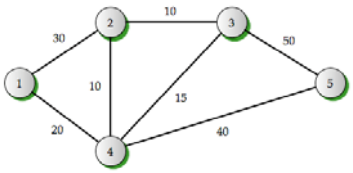
   Drop $k$ from $\mathcal{S}'$, i.e., $\mathcal{S}' = \mathcal{S}' \backslash \{k\}$.

   If $\mathcal{S}'$ is empty, stop.
4. Consider nodes in $\mathcal{S}'$ to update maximum bandwidth path, i.e., for $j \in \mathcal{S}'$

   $$\underline{B}_{ij}(t) = \max\{\underline{B}_{ij}(t), \min\{\underline{B}_{ik}, b^i_{kj}(t)\}\}. \qquad (2.7.1)$$

   Go to Step 3.

# Illustration (1)




- $S = \{1\}$, $S' = \{2, 3, 4, 5\}$
- $\underline{B}_{12} = 30$, $\underline{B}_{14} = 20$, $\underline{B}_{13} = \underline{B}_{15} = 0$
- $\text{Max}_{j \in S'} \underline{B}_{1j} = 30$ is attained at $j = 2$
  - Now, $S = \{1, 2\}$, $S' = \{3, 4, 5\}$
- Next compare:
  - $\underline{B}_{13} = \max\{\underline{B}_{13}, \min\{\underline{B}_{12}, b_{23}\}\} = 10$ // use 1-2-3
  - $\underline{B}_{14} = \max\{\underline{B}_{14}, \text{miin}\{\underline{B}_{12}, b_{24}\}\} = 20$ // stay on 1-2
  - $\underline{B}_{15} = \max\{\underline{B}_{15}, \min\{\underline{B}_{12}, b_{25}\}\} = \max\{0, \min\{30,0\}\} = 0$
- $\text{Max}_{j \in S'} \underline{B}_{1j} = \max\{\underline{B}_{13}, \underline{B}_{14}, \underline{B}_{15}\} = 20$, attained for j=4.
  - Now, $S = \{1, 2, 4\}$, $S' = \{3, 5\}$
- And so on

# Illustration (2`

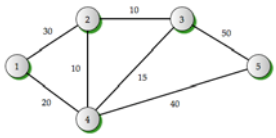



**TABLE 2.4** Iterative steps based on Algorithm 2.8.

| Iteration | List, $\mathcal{S}$ | $\underline{B}_{12}$ | Path | $\underline{B}_{13}$ | Path | $\underline{B}_{14}$ | Path | $\underline{B}_{15}$ | Path |
|---|---|---|---|---|---|---|---|---|---|
| 1 | {1} | 30 | 1-2 | 0 | – | 20 | 1-4 | 0 | – |
| 2 | {1, 2} | 30 | 1-2 | 10 | 1-2-3 | 20 | 1-4 | 0 | – |
| 3 | {1, 2, 4} | 30 | 1-2 | 15 | 1-4-3 | 20 | 1-4 | 20 | 1-4-5 |
| 4 | {1, 2, 4, 3} | 30 | 1-2 | 15 | 1-4-3 | 20 | 1-4 | 20 | 1-4-5 |
| 5 | {1, 2, 4, 3, 5} | 30 | 1-2 | 15 | 1-4-3 | 20 | 1-4 | 20 | 1-4-5 |

27

# Bellman-Ford-based Widest path routing

- Similar to additive version:

ALGORITHM 2.9 **Widest path algorithm, computed at node $i$ (Bellman–Ford-based).**

Initialize

$$\overline{B}_{ii}(t) = 0; \quad \overline{B}_{ij}(t) = 0, \quad \text{(for node } j \text{ that node } i \text{ is aware of).} \tag{2.7.2a}$$

For (nodes $j$ that node $i$ is aware of) do

$$\overline{B}_{ij}(t) = \max_{k \text{ directly connected to } i} \min \left\{ b_{ik}(t), \overline{B}^{i}_{kj}(t) \right\}, \quad \text{for } j \neq i. \tag{2.7.2b}$$

28

# Terminology

- Shortest-Widest Path
  - The widest path that has the least cost
- Widest-shortest path
  - A feasible path in terms of minimum cost
  - If there are several paths, one with maximum width is to be chosen

29

TABLE 2.5 Summary of notations used in this chapter

| Notation | Remark |
|---|---|
| $i$ | Source node |
| $j$ | Destination node |
| $k$ | Intermediate node |
| $\mathcal{N}$ | List of nodes in a network |
| $\mathcal{S}$ | Permanent list of nodes in the Dijkstra's algorithm (considered so far in the calculation) |
| $\mathcal{S}'$ | Tentative list of nodes in the Dijkstra's algorithm (yet to be considered in calculation) |
| $\mathcal{N}_k$ | List of neighboring nodes of node $k$ |
| $d_{ij}$ | Link cost between nodes $i$ and $j$ |
| $d_{ij}(t)$ | Link cost between nodes $i$ and $j$ at time $t$ |
| $\overline{D}_{ij}$ | Cost of the minimum cost path from node $i$ to node $j$ (Bellman–Ford) |
| $\overline{D}_{ij}^{(h)}$ | Cost of the minimum cost path from node $i$ to node $j$ when $h$ hops have been considered |
| $\overline{D}_{ij}(t)$ | Cost of the minimum cost path from node $i$ to node $j$ at time $t$ |
| $d_{kj}^{i}(t)$ | Link cost between nodes $k$ and $j$ at time $t$ as known to node $i$ |
| $\overline{D}_{kj}^{i}(t)$ | Cost of the minimum cost path from node $k$ to node $j$ at time $t$ as known to node $i$ |
| $\underline{D}_{ij}$ | Cost of the minimum cost path from node $i$ to node $j$ (Dijkstra) |
| $\widehat{D}_{ij/P}(t)$ | Cost of path $p$ from node $i$ to node $j$ (path caching) |
| $\underline{B}_{ij}(t)$ | Nonadditive cost (width) of the best path from node $i$ to node $j$ at time $t$ (Dijkstra) |
| $\widehat{B}_{ij}(t)$ | Nonadditive cost (width) of the best path from node $i$ to node $j$ at time $t$ (path caching) |
| $\mathcal{P}_{ij}$ | The list of cached path at node $i$ for destination $j$ |
| $H_{ij}$ | Next hop for source $i$ for destination $j$ |

30

# K-shortest paths algorithm

- When we need to generate the shortest path, the next shortest path, up to the K-th shortest path
- Naïve idea:
  - Run the shortest path algorithm
  - Delete each link one at a time, and re-run shortest path algorithm on the reduced graph
    - Pick the best from this set of paths
  - … and so on
  - (not efficient computationally)
- A sketch of a more efficient algorithm is in the next slide

31

ALGORITHM 2.10 ***k*-shortest paths algorithm.**

1. Initialize $k := 1$.
2. Find the shortest path $\mathcal{P}$ between source ($i$) and destination ($j$) in graph $\mathcal{G}$, using Dijkstra's Algorithm.

   Add $\mathcal{P}$ to permanent list $\mathcal{K}$, i.e., $\mathcal{K} := \{\mathcal{P}\}$.

   If $K = 1$, stop.

   Add $\mathcal{P}$ to set $\mathcal{X}$ and pair $(\mathcal{P}, i)$ to set $\mathcal{S}$, i.e., $\mathcal{X} := \{\mathcal{P}\}$ and $\mathcal{S} := \{(\mathcal{P}, i)\}$.
3. Remove $\mathcal{P}$ from $\mathcal{X}$, i.e., $\mathcal{X} := \mathcal{X} \backslash \{\mathcal{P}\}$.
4. Find the unique pair $(\mathcal{P}, w) \in \mathcal{S}$, and corresponding deviation node $w$ associated with $\mathcal{P}$.
5. For each node $v$, except $j$, on subpath of $\mathcal{P}$ from $w$ to $j$ ($sub_{\mathcal{P}}(w, j)$):

   Construct graph $\mathcal{G}'$ by removing the following from graph $\mathcal{G}$:

   (a) All the vertices on subpath of $\mathcal{P}$ from $i$ to $v$, except $v$.

   (b) All the links incident on these deleted vertices.

   (c) Links outgoing from $v$ toward $j$ for each $\mathcal{P}' \in \mathcal{K} \cup \{\mathcal{P}\}$, such that $sub_{\mathcal{P}}(i, v) = sub_{\mathcal{P}'}(i, v)$.

   Find the shortest path $\mathcal{Q}'$ from $v$ to $j$ in graph $\mathcal{G}'$ using Dijkstra's Algorithm.

   Concatenate subpath of $\mathcal{P}$ from $i$ to $v$ and path $\mathcal{Q}'$, i.e., $\mathcal{Q} = sub_{\mathcal{P}}(i, v) \oplus \mathcal{Q}'$.

   Add $\mathcal{Q}$ to $\mathcal{X}$ and pair $(\mathcal{Q}, v)$ to $\mathcal{S}$, i.e., $\mathcal{X} := \mathcal{X} \cup \{\mathcal{Q}\}$ and $\mathcal{S} := \mathcal{S} \cup \{(\mathcal{Q}, v)\}$.
6. Find the shortest path $\mathcal{P}$ among the paths in $\mathcal{X}$ and add $\mathcal{P}$ to $\mathcal{K}$, i.e., $\mathcal{K} := \mathcal{K} \cup \mathcal{P}$.
7. Increment $k$ by 1.
8. If $k < K$ and $\mathcal{X}$ is not empty, go to Step 4, else stop.

32

# Summary

- Covers shortest path routing and widest path routing
- Centralized vs. Distributed
- Bellman—Ford, Distance Vector, Dijkstra's algorithm for shortest paths
  - Their variation for widest paths
- K-shortest path algorithm
- Their use in different contexts will be discussed in subsequent chapters

33