CHAPTER 4

THE GENERAL SHORTEST PATH PROBLEM

In this chapter, we shall consider the shortest path problem in a general directed network. For a network $G = (N, A)$, we shall assume that the set of nodes $N$ have been numbered arbitrarily from 1 to $n$, every arc in $A$ is directed, and there are no self-loops or parallel arcs. An arc is written as $(i, j)$, indicating that it is directed from node $i$ to node $j$, and is associated with a length $d_{ij}$ (which may stand for cost, travel time, etc.). Note that we do not assume that $d_{ij} = d_{ji}$.

1. Acyclic Network

A network is called **acyclic** if it has no directed circuits. We shall see that for such networks there is a natural stage variable that increases along any path, and therefore allows a simple DP formulation for finding a shortest path.

**Exercise 4.1.** Prove that every acyclic network has at least one node that has only outward-pointing arcs.

**Theorem 4.1.** In an acyclic network $G = (N, A)$, there exists a numbering of the nodes so that if $(i, j) \in A$, then $i < j$.

**Proof.** We construct such a numbering as follows: Pick any node that has only outward-pointing arcs (by Exercise 1, at least one such node exists), and number it as node 1. Remove this node and the arcs incident with it. The remaining subnetwork is still acyclic. In the subnetwork number any node with only outward-pointing arcs as node 2 and then remove it. Continue in this manner until all the nodes have been numbered.

For any arc $(i, j) \in A$, node $j$ has an inward-pointing arc as long as node $i$ has not been removed in the above numbering process. This means node $i$ must be numbered before node $j$ and it follows that $i < j$. $\square$

Suppose that the nodes of an acyclic network have been numbered as above. A (forward) DP can be formulated to find the shortest paths from node 1 to all other nodes as follows:

(i) Define the optimal value function as

\[ f_i = \text{the length of the shortest path from node 1 to node } i. \]

(ii) Recurrence relation:

\[ f_i = \min_{j<i} [f_j + d_{ji}] \quad i = 2, 3, \ldots, n \]

(let $d_{ji} = \infty$ if $(j, i) \notin A$).

(iii) Define the optimal policy function as $p(i) = j$ where $j$ minimizes (1). Note that $p(i)$ denotes the node that immediately precedes node $i$ on the shortest path from node 1 to node $i$. 51
(iv) Boundary condition:

\[ f_1 = 0 \]

(v) Answers: \( f_i \) for \( i = 1, 2, \ldots, n \).

Notice that the optimal value function is a function of a stage variable only, because there is only one possible state at each stage.

**Theorem 4.2.** The optimal value functions \( f_i, i = 1, 2, \ldots, n \) are correctly given by (1) and (2).

**Proof.** We prove by induction on \( i \). Assume that \( f_j \) is correctly given by (1) and (2) for \( j = 1, 2, \ldots, i - 1 \). Consider node \( i \). Every arc entering node \( i \) must come from a node with a lower number. Hence the shortest path from node 1 to node \( i \) must have length \( f_j + d_{ji} \) for some \( j < i \). Such \( j \) must be chosen optimally as in (1). Hence \( f_i \) is correctly given by (1).

**Exercise 4.2.** (a) Give a backward DP formulation for the problem of finding the shortest path from all nodes to node \( n \) in an acyclic network.

(b) Show that the above DP requires approximately \( N^2 \) operations (including additions and comparisons but not the effort needed to number the nodes initially).

## 2. General Networks

In this section we consider shortest path problems for a general network which may contain directed circuits. In this case there may not be a natural ordering of the nodes as in Theorem 1.

We are usually interested in 3 kinds of shortest path problems:

- **Problem A:** The shortest path between a specified pair of nodes.
- **Problem B:** The shortest paths from a node to all other nodes.
- **Problem C:** The shortest paths between all pairs of nodes.

Since most of the algorithms solving Problems A and B are essentially the same, we shall discuss Problem B only.

**Problem B:**

Consider the problem of finding shortest paths from node 1 to all other nodes in a network where the nodes have been numbered arbitrarily. Before we give a DP formulation, let us first consider an algorithm developed by Dijkstra, which is applicable when all \( d_{ij} \geq 0 \).

The algorithm tries to find the shortest path to each of the nodes on the network in the following order: First, it searches for the nearest node to the given node 1 (i.e. the shortest path from node 1 to this node is shorter than or equal to the shortest path to any other node) and the corresponding shortest path. Then it searches for the 2nd nearest node and its shortest path, and then the 3rd one and so on. In searching for the \( k^{th} \) nearest node, it makes use of the observation that the \( k^{th} \) nearest node is adjacent with one of the \( j^{th} \) nearest node, for some \( j < k \) (since the shortest path to node \( k \) must pass through such a node \( j \)). It follows that if we have found the set \( T \) of the 1st, 2nd, \ldots, \((k - 1)^{th}\) nearest nodes, then
we can easily find the $k^{th}$ nearest node. For example, the $k^{th}$ nearest node is the node $i$ that minimizes

$$\min_{u \in T \cup \{1\}} \{l^*(u) + d_{ui}\}.$$

where $l^*(u)$ denotes the length of the shortest path from node 1 to node $u$.

**DIJKSTRA’S ALGORITHM:** For networks with non-negative arc lengths.

To ensure that the algorithm is carried out in a systematic manner, let us introduce a labeling procedure. For each node, we shall give it two kinds of labels, *temporary* and *permanent* labels. The permanent label $l^*(i)$ associated with a node $i$ stands for the length of the shortest path from node 1 to node $i$; while a temporary label $l(i)$ is either infinite or stands for the length of some path (not necessarily shortest) from node 1 to node $i$ (therefore $l(i)$ is an upper bound for the length of the shortest path to node $i$).

Dijkstra’s procedure can be given formally as follows:

Step 0: Set $l(i) = d_{1i}$ for $i = 2, 3, \ldots, n$. For convenience, we can take $d_{1i} = \infty$ if there is no arc joining node 1 and node $i$. Set $l^*(1) = 0$ and $T = \{1\}$, where $T$ denote the set of nodes given permanent labels.

Step 1: Pick $k$ such that $l(k) = \min\{l(i) : i \in \{1, 2, 3, \ldots, n\} \setminus T\}$.

Set $l^*(k) = l(k)$, i.e. give node $k$ a permanent label.

Set $T \leftarrow T \cup \{k\}$. Stop if $T = \{1, 2, 3, \ldots, n\}$.

Step 2: Let node $k$ be the node that has just received a permanent label in Step 1. Replace all temporary labels of the neighbors of node $k$ by the following rule:

$$l(i) \leftarrow \min[l(i), l^*(k) + d_{ki}]$$

Return to Step 1.

Notation: (1) We denote the length of a path by $|P|$. (2) For the convenience of proving the following theorem, we denote the set $T$ and the temporary labels $l(i)$ by $T_k$ and $l_k(i)$ after the $k^{th}$ iteration of Dijkstra’s procedure (i.e after the $k^{th}$ pass through steps 1 and 2) for $k = 1, 2, \ldots, n - 1$; and denote their initial values (assigned in step 0) by $T_0$ and $l_0(i)$.

Note: It is easy to see that for each $j \notin T_m$, $l_m(j)$ is either infinite or represents the length of a path from node 1 to node $j$ and takes the form $\min_{i \in T_m} \{l^*(i) + d_{ij}\}$.

**THEOREM 4.3.** After the $m^{th}$ iteration of the Dijkstra’s procedure for $m = 1, 2, \ldots, n - 1$:

(i) For each $i \in T_m$, $l^*(i)$ represents the length of the shortest path from node 1 to node $i$;
(ii) $T_m$ contains node 1 and the first $m$ nearest nodes from node 1.

**PROOF.** We use induction. Initially, $T_0 = \{1\}$ and $l^*(1) = 0$, and so (i) and (ii) hold. Assume that (i) and (ii) hold after $m - 1$ iterations. Let node $k$ be given a permanent label at the $m^{th}$ iteration, and let node $u$ be the $m^{th}$ nearest node from node 1. We want to show that the length of the shortest path from node 1 to node $u$ is equal to $l^*(k)$ (This will imply that node $k$ is also an $m$-th nearest node from node 1 and $l^*(k)$ represents the length of the shortest path from node 1 to node $k$. Let node $v$ be the node adjacent to $u$ on the shortest path $P$ from node 1 to node $u$. Clearly $v$ is either node 1 or a node nearer to node 1 than node $u$. Node $v$ must therefore be in $T_{m-1}$, and we have $|P| = (\text{length of shortest path from node 1 to node } v) + d_{uv} = l^*(v) + d_{uv}$. Since $l^*(k)$ represents the length of a path from node 1 to node $k$, we have $l^*(k) \geq |P|$. On the other hand, since $l^*(k) = \min_{i \in T_{m-1}, j \notin T_{m-1}} (l^*(i) + d_{ij})$, we have $l^*(k) \leq l^*(v) + d_{uv} = |P|$. Thus $l^*(k) = |P|$. \qed
**Remark 4.1.** After \( m - 1 \) iterations, if there are two nodes whose shortest paths from node 1 are of the same length and are both the \( m^{th} \) nearest node from node 1, the Dijkstra’s procedure will select one of these as the \( m^{th} \) nearest node. At the next iteration, the other node will be selected as the \((m + 1)^{st}\) nearest node.

**Example 4.1.** Consider the network shown in Figure 1 where the numbers beside the arcs are the arc lengths. The arcs are undirected, i.e. we assume that \( d_{ij} = d_{ji} \) = the number beside each arc. We shall put temporary labels inside each node and when the label becomes permanent, we shall add a star on the number. Whenever arcs are used in a shortest path, we shall use heavy lines to represent them.

![Network Diagram](image)

Step 0: All nodes receive temporary label \( l(i) = d_{1i} \) and the node \( V_1 \) gets permanent label \( l^*(1) = 0 \) as in Figure 2.

![Network Diagram with Temporary Labels](image)

Step 1: \( V_3 \) has minimum temporary label 2, so \( V_3 \) receives permanent label.

\[
l^*(3) = l(3) = 2.
\]

Step 2: The temporary labels of \( V_3 \)'s neighbors \( V_2 \) and \( V_5 \) are updated:

\[
\begin{align*}
l(2) &\leftarrow \min[l(2), l^*(3) + d_{32}] = 3 \\
l(5) &\leftarrow \min[l(5), l^*(3) + d_{35}] = 13
\end{align*}
\]

The result is shown in Figure 3.

![Network Diagram with Permanent Labels](image)

Step 1: \( V_2 \) receives permanent label.
Step 2: The temporary labels of \( V_2 \)'s neighbours \( V_7 \) and \( V_8 \) are updated.

Step 1: \( V_8 \) receives permanent label.
Step 2: The temporary label of \( V_8 \)'s neighbour \( V_4 \) is updated.
This is shown in Figure 4.

Step 1: \( V_4 \) receives permanent label.
Step 2: The temporary labels of \( V_4 \)'s neighbours \( V_6 \) and \( V_7 \) are updated.

Step 1: \( V_6 \) receives a permanent label.
Step 2: Temporary label of \( V_6 \)'s neighbour \( V_7 \) is updated.

Step 1: \( V_5 \) receives a permanent label.
Step 2: Temporary label of $V_7$ ($V_5$’s neighbour) is updated.

```
        4*     1       5*       6
   0* ---- 2 ----- 3* 4 ---- 6
      2       1     11       13
```

Step 1: $V_6$ receives a permanent label.
The final result is shown in Figure 6.

```
        4*     1       5*       6
   0* ---- 2 ----- 3* 4 ---- 6
      2       1     11       13
```

One question remains after we have found the lengths of the shortest paths from node 1 to the other nodes, i.e. $l^*(i)$ for $i = 2, 3, \ldots, n$: how do we trace the paths? Clearly if we know how to determine the heavy arcs in the Figure 1 - 5 (Exercise), then we can determine the last intermediate node on the shortest path to a node. In Figure 6, we have written down the last intermediate node as the second number on each node. With that, we can easily trace a shortest path.

Let us now compute the number of operations required by the Dijkstra’s procedure. A total of $n - 1$ iterations are required. At the $i^{th}$ iteration, $(n - 1) - i$ comparisons are required to determine the minimum temporary label in step 1, and at most $(n - 1) - i$ additions and comparisons are required to update the temporary labels in step 2. Therefore a total of at most $3n^2/2$ operations are required.

We now show that Dijkstra’s procedure can be given a DP formulation as follows:

1. Define the optimal value function $f_i(j)$ as follows: $f_i(j) =$ the length of the shortest path from node 1 to node $j$ when we are restricted to use paths such that all nodes preceding node $j$ belong to $T_i$, for $i = 0, 1, 2, \ldots, n - 1$ and $j = 2, 3, \ldots, n$. Here, $T_i$ denotes the set that contains node 1 and $i$ other nodes that are closest to node 1.
2. At stage $i$ ($i = 1, 2, \cdots, n - 1$) determine a node $k_i$ such that $k_i \notin T_{i-1}$ and $f_{i-1}(k_i) = \min_{j \notin T_{i-1}} f_{i-1}(j)$. Set $T_i = T_{i-1} \cup \{k_i\}$. Then the recurrence relation can be written as:

$$f_i(j) = \begin{cases} f_{i-1}(j) & \text{if } j \in T_i \\ \min\{f_{i-1}(j), f_{i-1}(k_i) + d_{k_i,j}\} & \text{if } j \notin T_i \end{cases}$$

3. Boundary conditions: $T_0 = \{1\}$, and $f_0(j) = d_{1j}$ for $j = 2, 3, \cdots, n$.
4. Answers: $f_i(k_i)$ for $i = 1, 2, \cdots, n - 1$.
5. Optimal policy function is defined as:

$p(i) =$ the immediate preceding node of node $i$ on the shortest path from node 1 to node $i$. 

2. GENERAL NETWORKS

\[ p(i) \text{ can be found when node } i \text{ is placed into } T_j \text{ at some iteration } j \text{ (i.e. at stage } j, \text{ node } i \text{ is chosen to be } k_j). \text{ We take } p(i) = m \text{ where } m \in T_{j-1} \text{ and } f_{j-1}(m) + d_{mi} = f_j(i). \]

It is clear that the above DP is just another way of writing down the Dijkstra’s procedure and therefore needs no further proof.

Let us now consider the case where the lengths \( d_{ij} \) are not necessarily non-negative. If a negative circuit (i.e. a directed circuit where the sum of the arc lengths is negative) exists in the network, then the shortest path problem will have an unbounded solution (i.e. the path lengths can be arbitrarily small if it goes around the negative circuit a large number of times). We give a DP formulation that can detect such circuits.

**FORD’S ALGORITHM:** For networks which may contain negative circuits.

1. Define the optimal value function as
2. \( f_i(k) = \text{the length of the shortest path from node 1 to node } k \text{ when } i \text{ or fewer arcs must be used, for } i = 1, 2, \cdots, n; k = 1, 2, \cdots, n. \)
3. Recurrence relation:
   \[
   f_i(k) = \min_{j \neq k} [f_{i-1}(j) + d_{jk}]
   \]
   for \( i = 1, 2, \cdots, n; k = 1, 2, \cdots, n. \)
   (One exception: For the case \( k = 1 \), we include \( j = k \) in the minimization of the r.h.s.).
4. Optimal policy function:
   \( p_i(k) \) = the node that precedes node \( k \) on the shortest path from node 1 to node \( k \) when \( i \) or fewer arcs must be used. It can be determined by recording the node \( j \) which minimizes the r.h.s. of the recurrence relation in (2).
5. Boundary conditions:
   \[
   f_0(k) = \begin{cases} 
   0 & \text{if } k = 1 \\
   \infty & \text{if } k = 2, 3, \cdots, n.
   \end{cases}
   \]
6. Answers: \( f_n(k) \) for \( k = 1, 2, \cdots, n. \)

If, for some \( j \), \( f_{j-1}(k) = f_j(k) \) for all \( k = 1, 2, \cdots, n \), the algorithm is said to have converged. We can stop and the answer is simply \( f_j(k) \) since no improvement can be made by further executing the recurrence relation. In this case, no negative circuit can exist. If no convergence occurs after \( n \) iterations, then it would mean that the shortest paths to some nodes have more than \( n - 1 \) arcs and that the network has a negative circuit (Since in a network without negative circuit, all shortest paths have \( n - 1 \) or fewer arcs).

Number of operations: Each iteration of the above algorithm requires \( n(n - 1) + 1 \) additions and \( n(n - 2) + 1 \) comparisons, and at most \( n \) iterations are required. Hence the total number of operations is at most \( 2n^3 \).

**Example 4.2.** We now apply the above procedure to the network given in Figure 7

![Network Diagram](image-url)
\[ f_0(1) = 0, \]
\[ f_0(2) = \infty, \]
\[ f_0(3) = \infty, \]
\[ f_0(4) = \infty. \]
\[ f_1(1) = \min[0 + 0, \infty + \infty, \infty + \infty, \infty + \infty] = 0, \quad p_1(1) = 1; \]
\[ f_1(2) = \min[0 + 2, \infty - 4, \infty + \infty] = 2, \quad p_1(2) = 1; \]
\[ f_1(3) = \min[0 + 5, \infty + 6, \infty + \infty] = 5, \quad p_1(3) = 1; \]
\[ f_1(4) = \min[0 + \infty, \infty + 3, \infty + 1] = \infty, \quad p_1(4) = 1,2, \text{or}, 3. \]
\[ f_2(1) = \min[0 + 0, 2 + \infty, 5 + \infty, \infty + \infty] = 0, \quad p_2(1) = 1; \]
\[ f_2(2) = \min[0 + 2, 5 - 4, \infty + \infty] = 1, \quad p_2(2) = 3; \]
\[ f_2(3) = \min[0 + 5, 2 + 6, \infty + \infty] = 5, \quad p_2(3) = 1; \]
\[ f_2(4) = \min[0 + \infty, 2 + 3, 5 + 1] = 5, \quad p_2(4) = 2. \]
\[ f_3(1) = \min[0 + 0, 1 + \infty, 5 + \infty, 5 + \infty] = 0, \quad p_3(1) = 1; \]
\[ f_3(2) = \min[0 + 2, 5 - 4, 5 + \infty] = 1, \quad p_3(2) = 3; \]
\[ f_3(3) = \min[0 + 5, 1 + 6, 5 + \infty] = 5, \quad p_3(3) = 1; \]
\[ f_3(4) = \min[0 + \infty, 1 + 3, 5 + 1] = 4, \quad p_3(4) = 2. \]
\[ f_4(1) = \min[0 + 0, 1 + \infty, 5 + \infty, 4 + \infty] = 0, \quad p_4(1) = 1; \]
\[ f_4(2) = \min[0 + 2, 5 - 4, 4 + \infty] = 1, \quad p_4(2) = 3; \]
\[ f_4(3) = \min[0 + 5, 1 + 6, 4 + \infty] = 5, \quad p_4(3) = 1; \]
\[ f_4(4) = \min[0 + \infty, 1 + 3, 5 + 1] = 4, \quad p_4(4) = 2. \]

Therefore, the shortest path from node 1 to node 4 is \( 1 - 3 - 2 - 4(p_4(4) = 2, \quad p_4(2) = 3, \quad p_4(3) = 1) \) and its length is 4. As an exercise, try to write the above computations in the form of a table.

**Problem C:** The shortest path between each pair of nodes.

Clearly one way of solving this problem is to employ the above algorithm \( n \) times to find the shortest paths from node 1 to all other nodes, then from node 2 to all other nodes, ...etc. However, there are more efficient methods for solving this problem.

Let us consider the following DP due to Floyd:

**FLOYD'S ALGORITHM:** for finding shortest path between every pair of nodes.

1. Define the optimal value function as:

\[ f_i(j, k) = \text{length of the shortest path from node } j \text{ to node } k \text{ when only paths with intermediate nodes belonging to the set of nodes } 1, 2, \cdots, i \text{ are allowed } (i = 0 \text{ corresponds to the path with only one arc } (j, k)); \quad i, j, k = 1, 2, \cdots, n. \]

\[
(4) \quad f_i(j, k) = \min \left\{ \begin{array}{l}
    f_{i-1}(j, k) \\
    f_{i-1}(j, i) + f_{i-1}(i, k)
\end{array} \right.
\]

for \( i = 1, 2, \ldots, n. \)
2. Boundary condition:

\[ f_0(j, k) = d_{jk} \]  

3. Answers: \( f_n(j, k) \) for \( j = 1, 2, \ldots, n \) and \( k = 1, 2, \ldots, n \).

4. Optimal Policy Function: \( p_i(j, k) \) = the first intermediate node on the shortest path from node \( j \) to node \( k \) using \( \{1, 2, \ldots, i\} \) as intermediate nodes. It can be determined as follows:

\[
p_i(j, k) = \begin{cases} 
  p_{i-1}(j, k) & \text{if first line of (4) is less than or equal to the second line} \\
  p_{i-1}(j, i) & \text{otherwise}
\end{cases}
\]

Notice by allowing \( j = k \) on the r.h.s. of (4), negative circuits can be detected when \( f_i(j, j) < 0 \) for some node \( j \).

**Exercise 4.3.** Justify the fact that \( f_i(j, k) \) is correctly given by (4) and (5).

**Exercise 4.4.** Find all shortest paths using Floyd’s procedure for the network shown in Figure 8. All arcs without arrow head are considered undirected, i.e. \( d_{ji} = d_{ij} \) = the number associated with the arc.

![Network Diagram](attachment:network.png)

**Exercise 4.5.** Determine the number of operations required by the Floyd’s procedure and compare it with the procedure of executing Ford’s algorithm \( n \) times, once for each node.

**Exercise 4.6.** Consider a general network with \( N \) nodes and \( d_{ij} \) that may be positive or negative. Define the value of a path to be the maximum \( d_{ij} \) along the path. Give an efficient procedure for finding the minimum value paths from node 1 to all other nodes.

**Exercise 4.7.** Suppose you wish to route a message from node 1 to node \( N \) in a network, where \( p_{ij} \) is the known probability that a message routed from node \( i \) to node \( j \) arrives at node \( j \). (Otherwise, the message is lost forever.) Give an efficient procedure for determining the path from node 1 to node \( N \) which has the maximal probability of arrival of the message at node \( N \).