Shortest path algorithms

Pramesh Kumar

IIT Delhi

September 30, 2025

Shortest Path

- Fundamental problem with numerous applications.
- Appears as a subproblem in many network flow algorithms.
- Easy to solve.

Shortest path problem

Definition (Path cost). The cost of a directed path $\pi = (i_1, i_2, ..., i_k)$ is the sum of cost of its individual links, i.e., $c^\pi = \sum_{i=1}^{k-1} t_{i,i+1}$.

Definition (Shortest Path Problem). Given $G(N, A)$, link costs $t : A \mapsto \mathbb{R}$, and origin $r \in N$, the shortest path problem (also known as single-source shortest path problem) is to determine for every non-source node $i \in N \backslash \{r\}$ a shortest cost directed path from node r.

OR

Definition (Shortest Path Problem). Given $G(N, A)$, link costs $t : A \mapsto \mathbb{R}$, and source $r \in N$, the shortest path problem is to determine how to send 1 unit of flow as cheaply as possible from r to each node $i \in N \backslash \{s\}$ in an uncapacitated network.

Types of shortest path (SP) problems

1. *Single-source shortest path*: SP from one node to all other nodes (if exists)
   1.1 with non-negative link costs.
   1.2 with arbitrary link costs.
2. *Single-pair shortest path* SP from between one node and another node.
3. *All-pairs shortest path* SP from every node to every node.
4. *Various generalizations of shortest path*:
   - Max capacity path problem
   - Max reliability path problem
   - SP with turn penalties
   - Resource-constraint SP problem
   - and many more

Lemma (Subpaths of shortest path are shortest paths)

Let $\pi = (r = i_1, ..., i_h = k)$ *be a shortest path from* r *to* k *and for* $1 \leq p \leq q \leq k$, *let* $\pi_{pq} = (i_p, ..., i_q)$ *be a subpath of* π *from* p *to* q. *Then,* π_{pq} *is a shortest path from* i_p *to* i_q.

Proof.

Decomposing path π into subpaths $\pi_{rp}, \pi_{pq},$ and π_{qk}, so that $c^{\pi} = c^{\pi_{sp}} + c^{\pi_{pq}} + c^{\pi_{qk}}$. Assume that $\pi^{'}_{pq}$ be a path such that $c^{\pi_{pq}} > c^{\pi^{'}_{pq}}$. Then, $\pi^{'} = \pi_{sp} + \pi^{'}_{pq} + \pi_{qk}$ has cost $c^{(\pi^{'})} = c^{\pi_{sp}} + c^{\pi^{'}_{pq}} + c^{\pi_{qk}} < c^{\pi}$, which contradicts that π is a shortest path from r to k. □

LP formulation for a single pair shortest path

$$x_{ij} = \begin{cases} 1 & \text{if } (i,j) \in A \text{ is on shortest path} \\ 0 & \text{otherwise} \end{cases}$$

$$\min_{\mathbf{x}} \sum_{(i,j)\in A} t_{ij} x_{ij}$$

$$\text{s.t.} \quad \sum_{j\in FS(i)} x_{ij} - \sum_{j\in BS(i)} x_{ji} = \begin{cases} 1 & \text{if } i = r \\ -1 & \text{if } i = s \\ 0 & \text{otherwise} \end{cases} \quad , \forall i \in N$$

$$1 \geq x_{ij} \geq 0, \forall (i,j) \in A$$

Remark. We can replace $x_{ij} \in \{0,1\}$ with $1 \geq x_{ij} \geq 0$ due to a property whose discussion we are skipping here.
Let's write its KKT conditions ...

Optimality conditions

Theorem

For every node $j \in N$, let $l(j)$ denote the cost of some directed path from source r to j. Then, $l(j)$ represent the shortest path costs if and only if they satisfy the following optimality conditions:

$$\boxed{l(j) \leq l(i) + t_{ij}, \forall (i,j) \in A} \qquad (\star)$$

Proof.

$\Longrightarrow$ Let $l(j)$ represent the SP cost labels for $j \in N$. Assume that they do not satisfy the $(\star)$. Then, some link $(i,j) \in A$ must satisfy $l(i) > l(j) + t_{ij}$. In this case, we can improve the cost of SP to node j by coming through node i, thereby contradicting the fact that $l(j)$ represents the SP label of node j.

□

Proof (contd.)

$\Longleftarrow$ Consider labels $l(j)$ satisfying $(\star)$. Let $(r = i_1, i_2 ..., i_k = j)$ be any directed path π from source r to node j. The conditions $(\star)$ imply that

$$\begin{aligned} l(j) = & l(i_k) \leq l(i_{k-1}) + t_{i_{k-1} i_k} \\ & l(i_{k-1}) \leq l(i_{k-2}) + t_{i_{k-2} i_{k-1}} \\ & \vdots \\ & l(i_2) \leq l_{i_1} + t_{i_1 i_2} = t_{i_1 i_2} \end{aligned}$$

Adding above inequations, we get $l(j) = l(i_k) \leq t_{i_{k-1} i_k} + t_{i_{k-2} i_{k-1}} + \cdots + t_{i_1 i_2} = \sum_{(i,j) \in \pi} t_{ij}$. Thus $l(j)$ is a LB on the cost of any directed path from r to j. Since $l(j)$ is the cost of some directed path from r to j, it is also an UB on the SP cost. Therefore, $l(j)$ is the shortest path cost from r to j. $\square$

Single-source shortest path

Assumptions

1. Network is directed
2. Link costs are integers
3. There exists a directed path from r to every other node (can be satisfied by creating an artificial link from s to other nodes)
4. The network does not contain a negative cycle.

Remark. For a network containing a negative cycle reachable from r, the above LP will be unbounded since we can send an infinite amount of flow along that cycle.

Can SP contain a cycle?

1. It cannot contain negative cycles.
2. It cannot contain positive cycles since removing the cycle produces a path with lower cost.
3. One can also remove zero weight cycle without affecting the cost of SP.

Label setting and label correcting algorithms

- Shortest path algorithms assign tentative distance label to each node that represents an upper bound on the cost of shortest path to that node.
- Depending on how they update these labels, the algorithms can be classified into two types:
    1. Label setting
    2. Label correcting
- Label setting algorithms make one label permanent in each iteration
- Label correcting algorithms keep all labels temporary until the termination of the algorithm.
- Label setting algorithms are more efficient but label correcting algorithms can be applied to more general class of problems.

Dijkstra's algorithm

A label setting algorithm

Input: Graph $G(N, A)$, link costs $\mathbf{t}$, and source r
Output: Optimal cost labels l and predecessors $pred$
procedure $\textsc{Dijkstra}(G, \mathbf{t}, r)$
    $SE = \{r\}$ ▷ Scan Eligible List
    $l(i) \leftarrow \infty, \forall i \in N\{r\}; l(r) \leftarrow 0$
    $pred(i) \leftarrow \texttt{NA}, \forall i \in N \backslash \{r\}; pred(r) \leftarrow 0$
    while $SE \neq \phi$ **do**
        Choose a node i with minimum $l(i)$ from SE
        for $j \in FS(i)$ **do**
            if $l(j) > l(i) + t_{ij}$ **then**
                $l(j) \leftarrow l(i) + t_{ij}$
                $pred(j) \leftarrow i$
                $SE \leftarrow SE \cup \{j\}$
            end if
        end for
    end while
end procedure

Label correcting algorithm

Input: Graph $G(N, A)$, costs $\mathbf{t}$, and source r
Output: Optimal cost labels l and predecessors $pred$
procedure $\textsc{LabelCorrecting}(G, \mathbf{t}, r)$
    $SE = \{r\}$ ▷ Scan Eligible List
    $l(i) \leftarrow \infty, \forall i \in N \backslash \{r\}; l(r) \leftarrow 0$
    $pred(i) \leftarrow \texttt{NA}, \forall i \in N \backslash \{r\}; pred(s) \leftarrow 0$
    while $SE \neq \phi$ **do**
        Remove an element i from SE
        for $j \in FS(i)$ **do**
            if $l(j) > l(i) + t_{ij}$ **then**
                $l(j) \leftarrow l(i) + t_{ij}$
                $pred(j) \leftarrow i$
                if j not in SE **then**
                    $SE = SE \cup \{j\}$
                end if
            end if
        end for
    end while
end procedure

Single pair shortest path

A^* algorithm

- This algorithm requires a heuristic cost $h(i)$ of reaching destination s from any node i. $h(i)$ should be a lower bound on the value of cost of reaching from i to s. In highway networks, $h(i)$ can be taken as the Euclidean distance between i and s divided by the highest speed possible in the network.
- The Dijkstra's algorithm can be slightly modified to convert it into A^* algorithm. Make the following changes in Line 8.
  Choose a node i *with minimum* $l(i) + h(i)$
  Stop the algorithm if $i = s$.

Shortest path in Directed Acyclic Graph (DAG)

Directed acyclic graphs and topological ordering

Definition (Directed acyclic graph (DAG)). A directed graph is DAG if does not contain any directed cycle.

Definition (Topological ordering). We say that a labeling $order$ of a graph is topological ordering if $\forall (i,j) \in A,$ we have $order(i) < order(j)$. A network containing directed cycle cannot be topologically ordered.

Conversely, a directed acyclic graph can be topologically ordered.

Input: Graph $G(N, A)$
Output: Topological ordering $order$ of N
procedure TopologicalOrdering(G)
    $inDegree(i) \leftarrow 0, \forall i \in N$
    $order(i) \leftarrow \texttt{NA}, \forall i \in N$
    $count \leftarrow 1$
    for $(i, j) \leftarrow A$ **do**
        $inDegree(j) \leftarrow inDegree(j) + 1$
    end for
    $Q \leftarrow \{n \in N : inDegree(n) = 0\}$
    while $Q \neq \phi$ **do**
        Remove "next" node i from Q
        $order(j) \leftarrow count$
        $count = count + 1$
        for $j \in FS(i)$ **do**
            $inDegree[j] \leftarrow inDegree[j] - 1$
            if $inDegree[j] == 0$ **then**
                $Q \leftarrow Q \cup \{j\}$
            end if
        end for
    end while
    if $count < |N|$ **then**
        G has cycle(s)
    else
        G is acyclic and return $order$
    end if
    return $order$
end procedure

Shortest path in acyclic networks

Remember that we can always order nodes in acyclic networks $G(N, A)$ such that $order(i) < order(j), \forall (i, j) \in A$ in $O(|A|)$ time.

Input: Graph $G(N, A)$, costs $\mathbf{t}$, and source r
Output: Optimal cost labels l and predecessors $pred$
procedure $\textsc{ShortestPathsDAG}(G, \mathbf{t}, s)$
    $l(i) \leftarrow \infty, \forall i \in N\{r\}; l(r) \leftarrow 0$
    $pred(i) \leftarrow \texttt{NA}, \forall i \in N\setminus\{s\}; pred(r) \leftarrow 0$
    $order \leftarrow \textsc{TopologicalOrdering}(G)$
    for each node i in $order$ **do**
        for $j \in FS(i)$ **do**
            if $l(j) > l(i) + t_{ij}$ **then**
                $l(j) \leftarrow l(i) + t_{ij}$
                $pred(j) \leftarrow i$
            end if
        end for
    end for
end procedure

Longest path in acyclic networks

Input: Graph $G(N, A)$, costs $\mathbf{t}$, and source r
Output: Optimal cost labels l and predecessors $pred$
procedure $\textsc{ShortestPathsDAG}(G, \mathbf{t}, s)$
    $l(i) \leftarrow -\infty, \forall i \in N\{r\}; l(r) \leftarrow 0$
    $pred(i) \leftarrow \texttt{NA}, \forall i \in N \backslash \{s\}; pred(r) \leftarrow 0$
    $order \leftarrow \textsc{TopologicalOrdering}(G)$
    for each node i in $order$ **do**
        for $j \in FS(i)$ **do**
            if $l(j) < l(i) + t_{ij}$ **then**
                $l(j) \leftarrow l(i) + t_{ij}$
                $pred(j) \leftarrow i$
            end if
        end for
    end for
end procedure

Suggested reading

1. BLU Book Chapter 2
2. AMO Chapter 4 and 5

Thank you!