Shortest Path

Pramesh Kumar

IIT Delhi

March 18, 2024

Shortest Path

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

Outline

Introduction

Single-source shortest path

All-pairs shortest path

Shortest path problem

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

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

OR

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

LP formulation

Primal

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

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

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

Dual

$$\max_{\mathbf{d}} (n-1)d_s - \sum_{i\in N\backslash\{s\}} d_i$$

$$\text{s.t.} \quad d_i - d_j \leq c_{ij}, \forall (i,j) \in A$$

$$d_i \text{ free }, \forall i \in N$$

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

Outline

Single-source shortest path

Assumptions

1. Network is directed
2. Link costs are integers
3. There exists a directed path from s 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 s, 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.

Shortest path trees

Definition (SP tree). A shortest path tree rooted at $s \in N$ is a directed subgraph $G^{'}(N^{'}, A^{'})$ where, $N^{'} \subseteq N$ and $A^{'} \subseteq A$ such that

1. $N^{'}$ is the set of nodes reachable from s
2. $G^{'}$ forms a tree rooted at s
3. $\forall i \in N^{'}$, the unique path from s to i in $G^{'}$ is a SP from s to i.

Remark. Shortest path are not unique neither are shortest path trees.

Lemma (Subpaths of shortest path are shortest paths)

Let $P = (s = i_1, ..., i_h = k)$ be a shortest path from s to k and for $1 \leq p \leq q \leq k$, let $P_{pq} = (i_p, ..., i_q)$ be a subpath of P from p to q. Then, P_{pq} is a shortest path from i_p to i_q.

Proof.

Decomposing path P into subpaths $P_{sp}, P_{pq},$ and P_{qk}, so that $c(P) = c(P_{sp}) + c(P_{pq}) + c(P_{qk})$. Assume that $P^{'}_{pq}$ be a path such that $c(P_{pq}) > c(P^{'}_{pq})$. Then, $P^{'} = P_{sp} + P^{'}_{pq} + P_{qk}$ has cost $c(P^{'}) = c(P_{sp}) + c(P^{'}_{pq}) + c(P_{qk}) < c(P)$, which contradicts that P is a shortest path from s to k. $\square$

Cost of shortest path

Lemma

Let $d(i)$ be the cost of shortest path from s to node $i \in N$. Then, a directed path P from s to k is a shortest path if and only if $d(j) = d(i) + c_{ij}, \forall (i,j) \in P$

Proof.

$\Longleftarrow$ Let $P = (s = i_1, ..., i_h = k)$ be a path from s to k such that $d(j) = d(i) + c_{ij}, \forall (i,j) \in P$. Then, cost of the path is

$$\begin{aligned} c(P) &= \sum_{(i,j) \in P} c_{ij} = c_{i_{h-1}, i_h} + ... + c_{i_1, i_2} \\ &= (d(i_h) - d(i_{h-1})) + (d(i_{h-1}) - d(i_{h-2})) + ... + (d(i_2) - d(i_1)) \\ &= d(i_h) = d(k) \end{aligned}$$

Therefore, $P(s = i_1, ..., i_h = k)$ is the shortest path from s to k.

$\Longrightarrow$ Let P be a shortest path from s to k and $d(k)$ is the cost of shortest path from s to k. Using previous lemma, since subpaths of shortest paths are also shortest paths, we have $d(j) = d(i) + c_{ij}, \forall (i,j) \in P$.

$\square$

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 $O(m + n)$ time.

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

Proposition

$\textsc{ShortestPathsDAG}$ *solves the shortest path algorithm on acyclic networks in* $O(m+n)$ *time.*

Proof.

Lines 4-5 take $O(n)$ time. Further, $\textsc{TopologicalOrdering}$ takes $O(m+n)$ time. The "for" loop of line 7 runs for each nodes. Then, it checks each link only once. Lines 9-11 takes $O(1)$ time. Therefore, the total running time is $O(m+n)$. □

Proposition

The labels $d(i), \forall i$ *computed by* $\textsc{ShortestPathsDAG}$ *on acyclic networks are optimal.*

Proof.

Use induction on i. □

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 $G(N, A)$, costs c, and source s
Output: Optimal cost labels d and predecessors $pred$
procedure DIJKSTRA(G, c, s)
    $S \leftarrow \phi; T \leftarrow N$
    $d(i) \leftarrow \infty, \forall i \in N\{s\}; d(s) \leftarrow 0$
    $pred(i) \leftarrow \texttt{NA}, \forall i \in N \backslash \{s\}; pred(s) \leftarrow 0$
    while $T \neq \phi$ **do**
        Choose a node i with minimum $d(i)$ from T
        $S \leftarrow S \cup \{i\}; T \leftarrow T \backslash \{i\}$
        for $j \in FS(i)$ **do**
            if $d(j) > d(i) + c_{ij}$ **then**
                $d(j) \leftarrow d(i) + c_{ij}$
                $pred(j) \leftarrow i$
            end if
        end for
    end while
end procedure

Running time of Dijkstra's algorithm

Two basic operations:

- Node selections: This is performed n times and each time, we need to scan the temporary labeled nodes. Total node selection time is $n + (n-1) + ... + 1 = O(n^2)$
- Label updates: This operation is performed $|FS(i)|$ times for each node i. Therefore, this operation requires $O(\sum_{i\in N} |FS(i)|) = O(m)$ time.
-

Therefore, total running time of the algorithm is $O(n^2 + m) = O(n^2)$ (for dense networks $m = \Omega(n^2)$). One can improve the running time on sparse networks and with efficient data structures.

Label correcting algorithm

- Special structure
  - Special topology (DAG) – Reaching algorithm
  - Non-negative costs – Label setting algorithm
- SP on a graph with negative cycles is a hard problem. Our aim is:
  - Either detect whether graph has negative cycles
  - If not, solve the problem

Optimality conditions

Theorem

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

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

Proof.

$\Longrightarrow$ Let $d(j)$ 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 $d(i) > d(j) + c_{ij}$. In this case, we can improve the cost of SP to node j by coming through node i, thereby contradicting the fact that $d(j)$ represents the SP label of node j.

□

Proof (contd.)

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

$$\begin{aligned} d(j) = & d(i_k) \leq d(i_{k-1}) + c_{i_{k-1}i_k} \\ & d(i_{k-1}) \leq d(i_{k-2}) + c_{i_{k-2}i_{k-1}} \\ & \vdots \\ & d(i_2) \leq d_{i_1} + c_{i_1 i_2} = c_{i_1 i_2} \end{aligned}$$

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

Label correcting algorithm

Input: Graph $G(N, A)$, costs c, and source s

Output: Optimal cost labels d and predecessors $pred$

procedure $\textsc{LabelCorrecting}(G, c, s)$

  $SEL = \{s\}$

  $d(i) \leftarrow \infty, \forall i \in N\{s\}; d(s) \leftarrow 0$

  $pred(i) \leftarrow \texttt{NA}, \forall i \in N \backslash \{s\}; pred(s) \leftarrow 0$

  while $SEL \neq \phi$ **do**

    Remove an element i from SEL

    for $j \in FS(i)$ **do**

      if $d(j) > d(i) + c_{ij}$ **then**

        $d(j) \leftarrow d(i) + c_{ij}$

        $pred(j) \leftarrow i$

        if j not in SEL **then**

          $SEL \leftarrow SEL \cup \{j\}$

        end if

      end if

    end for

  end while

end procedure

Running time

- Assume that data is integral, cost of each link is at most C, and no negative cycles.
- Each cost label $d(j)$ is bounded from above and below by $-nC$.
- The algorithm updates any label at most $2nC$ times (worst case every update reduces the label by 1 unit).
- Total number of distance label updates $= \sum_{i \in N} 2nC|FS(i)| = O(mnC)$.

Can we do better?

- We arrange the links in some order. Then, one iteration of the algorithm will check for every link (i, j) if it violates the optimality condition. If it does, then we update $d(j) = d(i) + c_{ij}$.
- We repeat above scanning of links for $n - 1$ iterations.
- This implies $O(mn)$ time bound which is strongly polynomial.
- This is also called Bellman-Ford algorithm.

Detecting negative cycles

- One can terminate when the label of any node falls below $-nC$.

Outline

All-pairs shortest path

Optimality conditions

Theorem

For every pair of nodes $(i, j) \in N \times N$*, let* $d[i, j]$ *represent the cost of some directed path from* i *to* j *satisfying* $d[i, i] = 0, \forall i \in N$ *and* $d[i, j] \leq c_{ij}, \forall (i, j) \in A$*. These costs represent the all-pairs shortest path costs if and only if*

$$\boxed{d[i, j] \leq d[i, k] + d[k, j], \forall i, j, k \in N}$$

.

Proof.

$\Longrightarrow$ We use contradiction. Let $d[i, j] > d[i, k] + d[k, j]$ for some $i, j, k \in N$. Then, the union of the shortest paths from i to k and k to j is a directed walk. Decompose that walk into a directed path P from i to j and some directed cycles (with non-negative costs). The cost of P is at most $d[i, k] + d[k, j] < d[i, j]$, which contradicts the optimal of $d[i, j]$.
$\Longleftarrow$ Similar to the one used for previous theorem. $\square$

Floyd-Warshall algorithm

Let $d_{ij}^{(k)}$ represent the cost of SP from i to j using the nodes only from $\{1, 2, \ldots, k-1\}$ as intermediate nodes. Clearly, $d_{ij}^{(n+1)}$ represents the SP cost from i to j.

$$d^{(k+1)}[i,j] = \min\left\{ \underbrace{d^{(k)}[i,j]}_{\text{SP not passing through } k}, \underbrace{d^{(k)}[i,k] + d^{(k)}[k,j]}_{\text{SP passing through } k} \right\}$$

Floyd-Warshall algorithm

```
procedure FloydWarshall(G, c)
    for (i, j) ∈ N × N do
        if (i, j) ∈ A then
            d[i, j] ← c_ij; pred[i, j] ← i
        else if i == j then
            d[i, i] ← 0; pred[i, j] ← NIL
        else
            d[i, j] ← ∞; pred[i, j] ← NA
        end if
    end for
    for k = 1 : n do
        for (i, j) ∈ N × N do
            if d[i, j] > d[i, k] + d[k, j] then
                d[i, j] ← d[i, k] + d[k, j]
                pred[i, j] ← pred[k, j]
            end if
        end for
    end for
end procedure
```

Runs in $O(n^3)$ time

Suggested reading

1. AMO Chapter 4 and 5

Origins of above algorithms

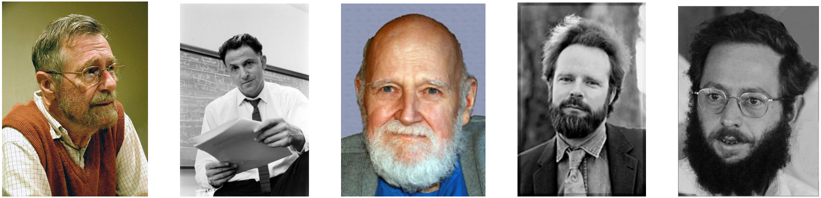

Figure: (From left to right) Edsger W. Dijkstra, Richard E. Bellman, Lester Randolph Ford Jr., Robert W Floyd, Stephen Warshall (Pictures source: Wiki, stanford.edu, and independent.com/)

Thank you!