5.1

Shortest chain Algorithms

Katta G. Murty, IOE 612 Lecture slides 5

Directed $G = (\mathcal{N}, \mathcal{A}, c)$. Find shortest chains in G. A special min cost flow problem of fundamental importance.

- Provides basic data for planning decisions transportation, routing & communication applications. Provides data for the design, capacity planning, & expansion of transportation and communication networks.
- Is the basis for CPM for planning decisions in project mgt.
- Has many applications in equipment replacement.
- Used to obtain travel time & distance charts between pairs of cities provided by AAAs, and the routes between pairs of locations provided on the web by web-servers.
- Appears as a subproblem in many other optimization algorithms.

Unboundedness of obj. func.: We consider **Unconstrained shortest chain Problem**. All chains from origin to destination feasible. The obj. func. (chain cost) is unbounded below if a chain exists in G from origin to destination, and there is a negative cost circuit.

Difficult cases: If G has a chain from origin to destination, and a negative cost circuit; and you want to find a **shortest simple chain** from origin to destination (this is a **constrained shortest chain problem**, because chains that are not simple are not feasible for this) it is a hard problem. Special case of finding the best extreme point sol. in an unbounded LP.

Transformations: **1. If G has an edge:** If G has an edge $(i; j)$ with cost $c_{ij} \geq 0$, replace it by a pair of arcs with same cost.

If $c_{ij} < 0$, this transformation does not work, as it immediately creates a negative cost simple circuit, making it hard to find shortest simple chains. In this case if there is no negative cost simple circuit in G, shortest simple chains can still be found efficiently in G using matching algos. (due to Roger Tobin, but technique of limited applicability).

So, we assume network directed.

2. If there are parallel arcs: Keep only cheapest among them & eliminate all others.

Assumption: G has no negative cost circuit: Algorithms contain procedures to check if assumption holds. Under assumption, cost of all circuits ≥ 0. So cost of any chain will never increase if any circuits in it are deleted. Hence if a chain exists from origin to destination, there will be a shortest chain which is simple.

All algorithms find shortest chains which are simple.

Fundamental property of shortest chains

If $\mathcal{P}$ is a shortest chain from 1 to n, and p, q are two nodes that appear on $\mathcal{P}$ in that order ; then the portion of $\mathcal{P}$ from p to q is a shortest chain from p to q.

Data structures for storing Shortest chains

Origin, destination specified: Can be stored as a sequence of nodes or arcs.

From an origin to all nodes in the network: Origin 1. When you put shortest chains from 1 to i, and that from 1 to j, suppose there is a cycle.

Can replace the portion from 1 to 4 in the chain from 1 to j, by that in chain from 1 to i. This leads to following union of shortest chains from 1 to i, and from 1 to j, no cycles.

Same way, can put together shortest chains from 1 to all other nodes, and eliminate all cycles in union. Union becomes a spanning tree with 1 as root, in which path from 1 to any other node is a shortest chain; tree is an outtree rooted at 1 called a **Shortest chain tree rooted at node 1**. Shortest chains stored by storing this tree using predecessor labels.

All shortest chains: Shortest chains between every pair of nodes. Stored by storing two $n \times n$ square matrices called **distance** and **label matrices**.

Distance matrix $D = (d_{ij})$, Label matrix $L = (\ell_{ij})$

d_{ij} = cost of shortest chain from i to j.

ℓ_{ij} = previous node to j in a shortest chain from i to j.

Entire shortest chain between any pair of nodes can be retrieved by looking up label matrix repeatedly.

Label matrix

	1	2	3	4	5	6
1	.	1	5	1	2	5
2	3	.	5	6	2	5
3	3	1	.	6	1	3
4	$\emptyset$	$\emptyset$	$\emptyset$	.	$\emptyset$	$\emptyset$
5	3	1	5	$\emptyset$	.	5
6	$\emptyset$	$\emptyset$	$\emptyset$	6	$\emptyset$	.

Distance matrix

	1	2	3	4	5	6
1	0	6.1	13.7	5.2	9.3	12.4
2	15.7	0	7.6	15.6	3.2	6.3
3	8.1	14.2	0	10.4	19.3	1.1
4	∞	∞	∞	0	∞	∞
5	12.3	18.4	4.4	12.4	0	3.1
6	∞	∞	∞	9.3	∞	0

LP formulations of unconstrained shortest chain problems

$G = (\mathcal{N}, \mathcal{A}, c, 1 = \text{origin}, n = \text{destination})$. $\Leftrightarrow$ sending one unit material from 1 to n across G at min cost, with $\ell = 0$, $k = \infty$, and $c =$ cost vector. So, LP formulation is:

$$\min \sum c_{ij} f_{ij}$$

$$\text{s. to.} \quad -f(i, \mathcal{N}) + f(\mathcal{N}, i) = \begin{cases} -1 & \text{if } i = 1 \text{ origin} \\ 0 & \text{if } i \neq 1 \text{ or } n \\ 1 & \text{if } i = n, \text{ destination} \end{cases}$$

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

$f_{ij} =$ no. of times chain traverses (i, j).

Why is capacity ∞ & not 1?

This model has a redundant constraint. We take it to be that corresponding to origin node 1. The dual problem is:

$$
\begin{aligned}
\min \pi_n - \pi_1 & & \\
\text{s. to} \quad \pi_j - \pi_i &\leq c_{ij} \quad \forall (i,j) \in \mathcal{A} \\
\pi_1 &= 0
\end{aligned}
$$

1. Every basic vector for the flow formulation consists of flow variables associated with arcs in a spanning tree of G.

2. A basic vector is feasible iff the unique path in the corresponding tree T from 1 to n is a chain.

   The BFS associated with s feasible spanning tree T is $f = (f_{ij})$ where (draw T as a rooted tree with 1 as rootnode):

$$
f_{in} = \begin{cases} 0 & \text{if } (i,j) \text{ not on predecessor path of } n \\ 1 & \text{if } (i,j) \text{ on the predecessor path of } n \end{cases}
$$

3. So each BFS for the flow formulation corresponds to a simple chain from 1 to n and vice versa.

4. If there is a negative cost circuit in G, the dual is infeasible. If there is a chain from 1 to n, and G contains a negative cost circuit, unconstrained shortest chain algos. will detect one such circuit & terminate with unboundedness conclusion.

To find Shortest simple chain for 1 to n in G - Solvable & hard cases

1. G has no negative cost circuit. Solvable in $O(n^2)$ or $O(m)$ or $O(nm)$ time.

2. G may have $-$ve cost circuit, but for every node i on a $-$ve cost circuit, either there is no chain from i to n, or there is no chain from 1 to i.

Let Y = set of all nodes j s. th. there is a chain from 1 to j, and a chain from j to n. In this case a shortest simple chain from 1 to n can be found efficiently by applying shortest chain algos. on the subnetwork induced by the set of nodes Y.

3. There is a node i in G that is both on a $-$ve cost circuit, and a chain from 1 to n. Finding shortest simple chain is hard in this case.

Won't putting a capacity of 1 on all arcs work in Case 3?

Unboundedness in presence of $-$ve cost circuits occurs due to traversal around such a circuit an ∞ times. So, won't putting a capacity of 1 on all arcs take care of this problem?

Bellman-Ford Eqs. for shortest chains from 1 to all other nodes

1st consider destination node n. Let $\hat{f} = (\hat{f}_{ij})$, $\hat{\pi} = (\hat{\pi}_i)$ be primal & dual opt. sols. for LP formulation. Opt. Conds. are:

Dual feasibility: $\hat{\pi}_j - \hat{\pi}_i \leq c_{ij} \quad \forall (i,j) \in \mathcal{A}$

Primal feasibility: The set $\{(i,j) : \hat{f}_{ij} = 1\}$ forms a chain from 1 to n

C.S. Conds.: $\hat{f}_{ij} = 1 \Rightarrow \hat{\pi}_j - \hat{\pi}_i = c_{ij}$

Equality of objectives: Opt. dual obj. value $= \hat{\pi}_n =$ opt. primal obj. value $= c\hat{f} =$ cost of shortest chain from 1 to n.

So, if $\bar{\pi} = (\bar{\pi}_i)$ satisfies dual feasibility, and if there exists a chain from 1 to n among set of arcs $\{(i,j) \in \mathcal{A} : \bar{\pi}_j - \bar{\pi}_i = c_{ij}\}$ then that chain is a shortest chain from 1 to n of cost $\bar{\pi}_n - \bar{\pi}_1$ (or $\bar{\pi}_n$ if $\bar{\pi}_1 = 0$), and $\bar{\pi}$ is dual opt.

Conversely define $\tilde{\pi}$ by

$$\tilde{\pi}_i = \begin{cases} 0 \quad \text{if } i = 1 \\ \infty \quad \text{if no chain from 1 to } i \\ \text{cost of shortest chain from 1 to } i \quad \text{otherwise} \end{cases}$$

1. $\forall (i,j) \in \mathcal{A}$, we get a chain of cost $\tilde{\pi}_i + c_{ij}$ from 1 to j by putting arc (i,j) at end of shortest path from 1 to i; this length $\geq \tilde{\pi}_j =$ cost of shortest chain from 1 to j; i.e., $\tilde{\pi}$ satisfies dual feasibility conds. mentioned above.

2. If $\mathcal{C}$ is any chain from 1 to p in the set of arcs $\{(i,j) \in \mathcal{A} : \tilde{\pi}_j - \tilde{\pi}_i = c_{ij}\}$, then its cost is sum $c_{ij} = \tilde{\pi}_j - \tilde{\pi}_i$ over arcs (i,j) in it $= \tilde{\pi}_p$; hence $\mathcal{C}$ is a shortest chain from 1 to p.

Hence $\tilde{\pi}$ satisfies the **Bellman - Ford eqs.**

$$\tilde{\pi}_1 = 0$$

$$\tilde{\pi}_j = \min\{\tilde{\pi}_i + c_{ij} : i \in B(j)\} \quad \forall j \neq 1$$

Nec. conds. for $\tilde{\pi}$ to be vector of shortest chain costs from 1 to other nodes.

Conversely if $\tilde{\pi}$ satisfies BF eqs. & there is a chain from 1 to i of cost $\tilde{\pi}_i$ then it is a shortest chain from 1 to i & all arcs (u, v) on it satisfy $\tilde{\pi}_v - \tilde{\pi}_u = c_{uv}$.

Methods for specified origin: Two classes of methods:

Label setting methods: SC tree grown one arc per step. At each stage, for each in-tree node, its predecessor path in reverse is a shortest chain from origin to it. Terminates when no more nodes can be included in tree.

Label correcting methods: Always maintains a spanning outtree rooted at origin. Changes it by one arc typically per step. Changes continue until tree becomes a SC tree.

LS Methods – Dijkstra's method

To find shortest chains in $G = (\mathcal{N}, \mathcal{A}, c, 1 = \text{origin node})$.

Assumption: $c \geq 0$.

Main theorem: $G = (\mathcal{N}, \mathcal{A}, c \geq 0, 1 = \text{origin})$. T SC tree, not spanning. X = set of in-tree nodes. For $i \in X$, π_i = cost of chain from 1 to i in T. $(p, q) \in$ Cut $(X, \bar{X})$ satisfies:

$$\pi_p + c_{pq} = \min\{\pi + c_{ij} : (i, j) \in (X, \bar{X})\}$$

Add arc (p, q) to T and define $\pi_q = \pi_p + c_{pq}$. This gives SC tree T' spanning nodes $X \cup \{q\}$.

Notes: Theorem used repeatedly until tree becomes spanning in $(n-1)$ steps. When $|X| = r$, effort to find next arc to add is $O(r(n-r))$. So, if implemented directly, overall complexity of method will be $\Sigma_{r=1}^{n} O(r(n-r)) = O(n^3)$.

Dijkstra reduced complexity to $O(n^2)$ by replacing cut examination with *setting and updating node labels called* **Temporary labels for out-of-tree nodes**. Method examines each arc precisely once.

Nodes in 3 states: **permanently labeled** (in-tree nodes); **temporarily labeled** (out-of-tree nodes one arc away from tree); **unlabeled** (other out-of-tree nodes).

Label on node i of form $(P(i), d_i)$ where:

$P(i) =$ predecessor index of node i for in-tree nodes, for labeled out-of-tree nodes it is the previous node to i on a shortest chain from 1 to i using only in-tree nodes as intermediate nodes.

$d_i =$ for in-tree nodes it is the cost of shortest chain from 1 to i, for labeled out-of-tree nodes it is cost of shortest chain from 1 to i using only in-tree nodes as intermediate nodes.

Once node becomes permanently labeled, it is in-tree, and its label will never change. Each step permanently labels one more node, so method takes $\leq n$ steps. Denote:

$X =$ set of permanently labeled node

$Y =$ set of temporarily labeled nodes

$N =$ set of unlabeled

Labels on nodes in Y updated in each step. One node moves from Y to X in each step, the one with smallest distance index in Y.

Dijkstra's method

Root Tree at origin: Permanently label 1 with $(\emptyset, 0)$. Temporarily label each $j \in A(1)$ with $(1, c_{1j})$. $X = \{1\}$, $Y = A(1)$, $N = \mathcal{N} \backslash (X \cup Y)$.

Tree growth step: If $Y = \emptyset$ go to termination step.

If $Y \neq \emptyset$, make label on i permanent. Move i from Y to X. If label on i is $(P(i), \pi_i)$, $(P(i), i)$ is new arc included in SC tree in this step.

> $\forall j \in Y$ let d_j be its distance index. If $(i, j) \in \mathcal{A}$ and $d_j > \pi_i + c_{ij}$ change temp. label on j to $(i, \pi_i + c_{ij})$.
>
> Temp. label each $j \in N \cap A(i)$ with $(i, \pi_i + c_{ij})$ and move all such j from N to Y.

If $X \neq \mathcal{N}$ repeat this tree growth step.

Termination step: We have SC tree. If $X \neq \mathcal{N}$, no chain from 1 to any node in N in G. Terminate.

Example

What if $c \ngeq 0$?

Theorem: If $c \geq 0$ method gives SC tree with complexity $O(n^2)$.

BrFS method is special case of this method for $c_{ij} = 1 \forall (i, j) \in \mathcal{A}$.

If shortest chains to only a subset of nodes are needed, method terminates when all those nodes are permanently labeled.

LC Methods for a specified origin

Work for general c, so no need to assume $c \geq 0$.

These are variants of primal simplex on LP formulation. Every basis is a spanning tree, and a pivot step exchanges an out-of-tree arc with an in-tree arc in its funda. cycle. Maintain a spanning outtree rooted at origin. Each iteration, labels on one or more nodes change.

Initial Spanning outtree selection: $\forall j \neq 1$ if $(1, j) \notin \mathcal{A}$ introduce artificial arc $(1, j)$ with cost c_{1j} = a large +ve no., say $= 1 + n(\max\{|c_{pq}| : (p, q) \in \mathcal{A}\})$.

Then set T_0 = spanning outtree determined by arcs $\{(1, j) : j \neq 1\}$.

Node Labels maintained by algos.: Of form $(P(i), d_i)$ where:

$P(i)$ = predecessor index of node i in current tree

d_i = distance index of i (will equal π_i = cost of present chain from 1 to i if step *Correcting distance index of descendents* carried out in each iteration; $d_i \geq \pi_i$ otherwise).

$E = \{(P(i), i) : i \neq 1\}$ (will equal set of arcs in present spanning outtree until algo. detects a $-$ve cost circuit; from that time the node labels and E will not represent a spanning outtree, instead they will represent some trees (not spanning) + one or more $-$ve cost circuits).

Algos. can terminate two ways: (1) with SC Tree (happens when distance indices satisfy dual feasibility); (2) with a $-$ve cost circuit.

In all these algos. artificial arcs eliminated once they become out-of-tree.

Classical primal method for specified origin

Main source of all LC methods. Labels are actually $(P(i), \pi_i)$.

Initialization: Start with T_0. Label 1 with $(\emptyset, 0)$, and all $i \neq 1$ with $(1, c_{1i})$.

General iteration: Let $(P(i), \pi_i)$ be present node labels.

1 : **Select incoming arc:** Select $(i, j) \in \mathcal{A}$ violating dual feasibility, i.e., satisfying $\pi_j > \pi_i + c_{ij}$.

If no such arc, TERMINATE, PRESENT OUTTREE IS AN SC TREE.

If such arc selected, let $\delta = \pi_j - \pi_i - c_{ij} > 0$; and $D_j =$ set of descendents of j in present tree.

2 : **Ancestor checking:** Check whether j is an ancestor of i.

If it is, arc (i, j) together with the portion of predecessor path of i between i and j is a $-$ve cost circuit of cost $-\delta$, TERMINATE.

3 : **Label correction:** Change label on j to $(i, \pi_i + c - ij)$. It replaces in-tree arc $(P(j), j)$ with (i, j), and reduces cost of chain to j by δ.

4 : **Correcting distance index of descendents:** Change π_p to $\pi_p - \delta \; \forall p \in D_j$.

Go to next iteration.

Examples:

Finiteness proof:

Complexity: Depends on rule used to select incoming arc. Can vary from polynomial time to exponential time.

Rules for selecting incoming arc: Takes $O(m)$ effort if carried out by examining all arcs.

Efficient implementations use **Branching out of node** i (examining arcs in forward star of i for dual feasibility) for i in a **List** (set of candidate nodes for branching out, maintained).

Ancestor checking: Adds $O(n)$ effort per iteration.

Eliminated if known that no $-$ve cost circuits exist. Even when not known, some implementations eliminate it. If (i, j) entered & j ancestor of i, E has $-$ve cost circuit thro' j from then on.

Correcting distance index of descendents: To get D_j efficiently, PIs not adequate, need other tree labels. So, some implementations do not carry this step. Distance labels will get corrected before termination.

Bellman-Ford-Moore (BFM) LC Algo.

Can be interpreted as a recursive (DP) or **successive approximation approach** to solve BF eqs. In $r+1$th iteration, obtains $r+1$th order approx. π^{r+1} from rth.

DEFINITION: π_j^r = distance index of j at end of rth iteration = cost of a shortest chain from 1 to j with $\leq r$ arcs.

Iteration 1: T_0 is initial outtree. Label 1 with $(\emptyset, 0)$ and $i \neq 1$ with $(1, c_{1i})$. $\pi_j^1 = c_{1j}\ \forall j$.

Iteration $r+1$**:** Let $(P(i), \pi_i^r)$ be label on i at end of iteration r. $\forall j \in \mathcal{N}$ compute:

$$\pi_j^{r+1} = \min\{\pi_j^r, \pi_i^r + c_{ij} \ \ \text{over} \ \ i \in B(j)\}$$

and let:

$$u_j = \begin{cases} P(j) & \text{if minimum above is } \pi_j^r \\ \text{an } i \in B(j) \text{ attaining min above} & \text{otherwise} \end{cases}$$

If $\pi_j^{r+1} = \pi_j^r\ \forall j \in \mathcal{N}$ **Stability attained**, present labels define an SC tree, TERMINATE.

Otherwise, $\forall j \in \{i : \pi_i^{r+1} < \pi_i^r\}$ change label to (u_j, π_j^{r+1}), and go to next iteration if $r+1 \leq n-1$. In this case if $r+1 = n$, a $-$ve cost circuit exists among present E, TERMINATE.

- if no $-$ve cost circuits, stability will be attained before $(n-1)$th iteration.
- if stability not attained after n iterations, E must contain a $-$ve cost circuit.
- overall complexity $O(nm)$.
- what if some artificial arcs $(1, i)$ remain in final SC tree?

FIFO LC Algo.

Primal algo. with branching out operation. List maintained as a Q with FIFO discipline. Ancestor checking, correcting distance index on descendents, not carried out. Complexity $O(nm)$.

Iteration 1: Begin with labels for T_0. List $= \{1\}$.

General iteration: Select the node for branching out from top of list, and continue until list becomes $\emptyset$.

During iteration arrange all nodes whose labels have changed in another Q called **Next list** according to one of following disciplines:

> FIFO/NO MOVE: If j not in next list, insert it at bottom. If j already in next list, leave it in current position.
>
> FIFO/MOVE: If j not in next list, insert it at bottom. If j already in next list, move it to bottom position.

When list becomes $\emptyset$, if next list $= \emptyset$, present labels define an SC tree, TERMINATE. Otherwise if next list $\neq \emptyset$; look for a $-$ve cost circute in E if iteration count n, or if iteration count $< n$

make next list the new list and go to next iteration.

Dynamic Breadth First Search (DBFS) LC Algo.

Define a_j = **Label Depth** of node j = no. of arcs in present chain from 1 to j.

a'_i = **label depth index** of $i \leq a_i \ \forall i$ always.

Labels of form $(P(i), d_i, a'_i)$. Correction on descendents not carried out, so $a'_i \leq a_i$, but will be correct at termination if an SC tree is obtained.

a'_i can only increase during algo. Like BrFS, this method in iteration r branches out only those nodes whose LDI is r.

For each h let $n(h)$ = no. of nodes j for which $a'_j = h$.

Iteration 0: Start with labels for T_0. $a'_1 = 0, a'_j = 1 \quad \forall j \neq 1$. List $= \mathcal{N} \backslash \{1\}$, Next list $= \emptyset$. $n(0) = 1, \quad n(1) = n - 1, \quad n(h) = 0 \quad \forall h > 1$.

Iteration r: **1. Select a node from list to branch out:** If list $= \emptyset$ go to 3. If list $\neq \emptyset$ delete a node i from list to branch out. Let label on i be $(P(i), d_i, a'_i)$. If $a'_i = r$ go to 2. Otherwise repeat this step.

2. Branching out of i**:** $\forall j \in A(i)$ do:

> Let label on j be $(P(j), d_j, a'_j)$. If $d_j \leq d_i + c_{ij}$ continue. If $d_j > d_i + c_{ij}$ change PI and DI of j to i, $d_i + c_{ij}$ respectively; and if $a'_j \neq r + 1$ subtract 1 from $n(a'_j)$.
>
> If $n(a'_j)$ is now 0, a $-$ve cost circuit identified, find it by tracing predecessor path of j until a node repeats, TERMINATE. Otherwise change a'_j to $r + 1$, add 1 to $n(r + 1)$, include j in next list.

Return to 1.

3. Set up for next iteration: If next list = $\emptyset$, present labels define a SC Tree, TERMINATE. Otherwise, if $r = n$ look for a $-$ve cost circuit in E and TERMINATE, or if $r < n$ make list = next list, next list = $\emptyset$, go to next iteration.

Notes: Consider no $-$ve cost circuits. Let π_j^* denote length of shortest chain from 1 to j, and b_j the smallest no. of arcs in a shortest chain from 1 to j. Let $L(r) = \{j : b_j = r\}$.

At start of iteration r list only consists of nodes with $a_i' = r$. Also, node i is branched out in iteration r only if a_i' remains $= r$ when algo. tries to select it.

At start of iteration r, $L(r) \subset$ list. And $d_i = \pi_i^* \ \forall i \in L(r)$. Also, in this iteration all nodes in $L(r)$ are branched out.

So, in this case algo. terminates with SC tree after $\leq (n-2)$ iterations. Each iteration needs $O(m)$ effort. So overall complexity $O(nm)$.

On networks with n = 5000, m= 60,000, method takes 4 seconds on a SUN 3 workstation.

Acyclic Shortest chain Algo.

$G = (\mathcal{N}, \mathcal{A}, c = (c_{ij}), 1 = \text{specified origin})$, acyclic with acyclic numbering of nodes.

If origin is $i \neq 1$, no chain from i to j $\forall$ $j < i$. So all nodes $j < i$ & arcs incident at them can be deleted. In remaining network nodes can be renumbered beginning with 1 for node i. So WLOG assume 1 is origin.

G has no circuits, so no question of $-$ve cost circuits. Following recursive (DP) algo of complexity $O(m)$ finds SC tree rooted at 1. Nodes are labeled in specific order 1, ..., n. All labels assigned are permanent.

Step 1: Label 1 with $(\emptyset, 0)$ rooting the tree at 1.

General step r: When we come here, we would have already labeled i, say with $(P(i), \pi_i)$ $\forall i = 1$ to $r - 1$. Find

$$\pi_r = \min\{\pi_i + c_{ir} : i \in B(r)\}$$

If $B(r) = \emptyset$, we define $\pi_r = \infty$, and there is no chain from 1 to r. Otherwise, let $P(r)$ be an i that attains min above, label r

with $(P(r), \pi_r)$.

If $r = n$, labels define an SC tree rooted at 1, spanning all the nodes that can be reached from 1 by a chain, TERMINATE. If $r < n$, go to next step.

EXAMPLE:

Matrix methods for all shortest chains

To find shortest chains between every pair of nodes in $G = (\mathcal{N}, \mathcal{A}, c)$. For any $i \neq j$ if $(i, j) \notin \mathcal{A}$, introduce artificial arc (i, j) with large positive cost. These methods terminate with either a $-$ve cost circuit, or all shortest chains.

Inductive Algo.

Due to Dantzig. Takes n steps. In rth step, we have all shortest chains in partial network induced by $\{1, \ldots, r\}$. Step $r+1$ brings node $r+1$ into set of included nodes.

Step 1: Begin with partial network of node 1.

General step $r+1$: Let $L^r = (L^r_{ij} : i, j = 1 \text{ to } r)$, $d^r = (d^r_{ij} : i, j = 1 \text{ to } r)$ be label & distance matrices at end of Step r.

For bringing node $r+1$ do computations for updating the two matrices in following order:

	To 1 2 ... r	$r+1$
from 1 $\vdots$ r	4	1
$r+1$	2	3

$$d^{r+1}_{i,r+1} = \min\{c_{i,r+1}; \quad d^r_{ij} + c_{j,r+1} : \quad j = 1 \text{ to } r,\ j \neq i\}$$

$L^{r+1}_{i,r+1} = i$ if above min is $c_{i,r+1}$; or a j that attains min above.

$$d^{r+1}_{r+1,i} = \min\{c_{r+1,i}; \quad c_{r+1,j} + d^r_{ji} : \quad j = 1 \text{ to } r,\ j \neq i\}$$

$L^{r+1}_{r+1,i} = i$ if above min is $c_{r+1,i}$; or a j that attains min above.

$$d^{r+1}_{r+1,r+1} = \min\{0; \quad d^r_{r+1,j} + c_{j,r+1} : \quad j = 1 \text{ to } r\}$$

$L^{r+1}_{r+1,r+1} = r+1$ if above min is 0.

If above min < 0, let p be a j attaining the min above. Then by combining the shortest chain from $r+1$ to p obtained above, with the shortest chain from p to $r+1$ obtained above, we get a $-$ve cost circuit, TERMINATE.

If $d^{r+1}_{r+1,r+1} = 0$, for $i, j = 1$ to r find:

$$d^{r+1}_{i,j} = \min\{d^{r}_{ij}; \quad d^{r+1}_{i,r+1} + d^{r+1}_{r+1,j}\}$$

$L^{r+1}_{ij} = L^{r}_{ij}$ if above min is d^{r}_{ij}, $L^{r+1}_{r+1,j}$ otherwise.

$L^{r+1} = (L^{r+1}_{ij})$, $d^{r+1} = (d^{r+1}_{ij})$ are new label and distance matrices.

If any diagonal entries in d^{r+1} are < 0, say d^{r+1}_{jj}, then circuit containing j identified using labels in L^{r+1} is a $-$ve cost circuit, TERMINATE.

Otherwise, if $r + 1 = n$, the label & distance matrices give shortest chains & their costs, TERMINATE. If $r + 1 < n$ go to next step.

EX. Prove that distance matrix satisfies triangle ineq.

EX. Prove algo. valid, & derive its complexity.

Floyd - Warshall Algo.

$G = (\mathcal{N}, \mathcal{A}, c)$, nodes 1, ..., n.

Definition: on any simple chain, nodes other than origin, destination called **Intermediate nodes.**

Only simple chains not containing intermediate nodes are those with only one arc.

n steps. L^r, d^r are label, distance matrices at end of Step r, representing:

d^r_{ij} = cost of shortest chain from i to j s. to constraint that all intermediate nodes on it are from $\{1, \ldots, r\}$ (i, j may not be from this set).

Triangle (or Triple) Operation : For any pair of nodes i, j and fixed node $r + 1$,

$$d^{r+1}_{ij} = \min\{d^r_{ij}, d^r_{i,r+1} + d^r_{r+1,j}\}$$

$L^{r+1}_{ij} = L^r_{ij}$ if $d^{r+1}_{ij} = d^r_{ij}$; $L^r_{r+1,j}$ otherwise.

F W Algo.

Step 0: L^0, d^0 defined by $L^0_{ij} = i$, $d^0_{ij} = c_{ij}$

General Step $r+1$: Let L^r, d^r be the matrices at end of Step r. Perform triple operations $\forall\ i, j \in \mathcal{N}$ and $r+1$. Let L^{r+1}, d^{r+1} be resulting matrices.

If any $d^{r+1}_{ii} < 0$, the circuit obtained by putting together present chains from i to $r+1$ & $r+1$ to i is a $-$ve cost circuit, TERMINATE.

If $d^{r+1}_{ii} = 0\ \forall\ i \in \mathcal{N}$, & $r+1 = n$, present chains are shortest, TERMINATE. If $r+1 < n$ go to next step.