Note on Network Flow Optimization

Zepeng CHEN
The HK PolyU

Date: January 11, 2023

1 Basic Algorithm and Theorem in Network

1.1 Complexity Analysis

1.2 Searching Algorithm

Search algorithm is to identify all nodes that can be reached by direct path. It is easy to see the search algorithm runs in $O(m+n) = O(m)$ time.

```
Begin
    Unmark all nodes in N;
    Mark node s; pred(s) = 0;
    next:=1; LIST:={s};
    While LIST≠ ϕ doo
    Begin
        Select a node i from LIST;
        If node i is incident to an admission arc (i,j), then
        Begin
            Mark node j;
            pred(j)=i;
            next:=next+1
            order(j):=next
            add j to LIST
        End
        Else delete node i from LIST
    End
End
```

1.3 Breath-first Search

Maintain the LIST as a queue, select nodes from the front of LIST and add them to the rear.

Lemma 1.1 (Breadth First Search theorem)

The breadth first search tree is the "shortest path tree", that is, the path from s to j in the tree has the fewest possible number of arcs.

Note on *Note that the shortest path tree here does not clarify the distance, i.e., each arc's distance is 1.*

1.4 Depth-first Search

Maintain the LIST as a stack, select nodes from the front of LIST and add them to the front. A depth-first order also satisfies the following properties,

- If node j is a descendant of node $i \neq j$, then order(j)>order(i).
- All the descendants of any node are ordered consecutively.

1.5 Acyclic Identification

Definition 1.1 (Topological Ordering)

The labeling of a graph is a topological ordering if every arc joins a lower-labeled node to a higher-labeled node. That is, for every $(i, j) \in A$, order(i)<order(j).

Proposition 1.1 (Unique Topological Ordering)

x

Proposition 1.2 (Topological ordering and acyclic)

A network is acyclic iff it possesses a topological ordering of its nodes.

Below is the *Topological sorting* algorithm to identify if the network is acyclic and give a topological ordering.

```
Begin
    For all i ∈ N, do indegree(i)=0;
    For all (i,j) ∈ A, do indegree(j)+=1;
    LIST:=ϕ; next:=0;
    For all i ∈ N do
    If indegree(i)=0, then LIST=LIST∪ {i};
    While LIST≠ ϕ do
    Begin
        Select a node i from LIST and delete it;
        next:=next+1; order(i) :=next;
        For (i,j) ∈ A, do
        Begin
            indegree(j)-=1;
            If indegree(i)=0, then LIST=LIST∪ {j};
        End
```

```
    End
    If next<n, then the network contains a cycle;
    Else it is acyclic, and the labeling is a topological ordering.
End
```

Proposition 1.3 (Adjacency matrix and acyclic)

A directed graph G is acyclic iff we can renumber its nodes so that its node-node adjacency matrix is a lower triangular matrix.

1.6 Flow Decomposition

Lemma 1.2 (Flow decomposition theorem 1)

Let $f \geq 0$ be a nonzero circulation. Then, there exist simple circulations $f^1, \ldots, f^k$, involving only forward arcs, and positive scalars $a_1, \ldots, a_k$, such that

$$f = \sum_{i=1}^{k} a_i f^i$$

Furthermore, if f is an integer vector, then each a_i can be chosen to be an integer.

Lemma 1.3 (Flow decomposition theorem 2)

Every path and cycle flow has a unique representation of non-negative arc flows. Let $f(p), p \in P$ and $f(w), w \in W$ be the path and cycle flows. Let $\delta_{ij}(p) = 1$ if $(i, j) \in p$ and 0 otherwise, $\delta_{ij}(w) = 1$ if $(i, j) \in w$ and 0 otherwise. Then,

$$x_{ij} = \sum_{p \in P} \delta_{ij}(p) f(p) + \sum_{w \in W} \delta_{ij}(w) f(w)$$

Conversely, every non-negative arc flow can be represented as a path and cycle flow (though not necessarily unique) with the following two properties:

- *Every directed path with positive flow connects a supply node to an excess node.*
- *At most n+m paths and cycles have non-zero flow. Out of these, at most m cycles have non-zero flow.*

Proof Note that each iteration we can construct a loop or a path to eliminate a node or an arc. And there are $n + m$ nodes and arcs, thus, it needs at most $n + m$ non-zero loop or path for iteration. And each time we construct a non-zero loop, we can remove an arc, thus, there are at most m non-zero loop we can construct. ■

There is also an algorithm to do *flow decomposition*.

```
x
```

2 Minimum Cost Flow Problem

In this problem, $b(i) > 0$ is a supply node, $b(i) < 0$ is a demand node. If a flow satisfies these constraints, it will be called a *feasible flow*.

$$\begin{aligned} \min \quad & \sum_{(i,j)\in A} c_{ij}x_{ij} \\ \text{s.t.} \quad & \sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = b(i), i \in N \quad \text{(Flow balance constraint)} \\ & l_{ij} \leq x_{ij} \leq u_{ij}, (i,j) \in A \quad \text{(Capacity constraint)} \end{aligned} \tag{1}$$

By summing the *Flow balance constraint*, we obtain the assumption:

$$\sum_{i\in N} b(i) = 0$$

We also have a matrix form like this, where N is the node-arc incidence matrix.

$$\begin{aligned} \min \quad & cx \\ \text{s.t.} \quad & Nx = b \quad \text{(Flow balance constraint)} \\ & l \leq x \leq u \quad \text{(Capacity constraint)} \end{aligned} \tag{2}$$

Note that follow problems are special variants:

- Shortest path problem,
  - If we only want the solution from node s to node t, then set $b(s) = 1, b(t) = -1$ and $b(i) = 0$.
  - If we want all shortest path to node i, then set $b(s) = n - 1$ and $b(i) = -1 \forall i \neq s$.
- Maximum flow problem (Min cut), here we set $b(i) = 0 \forall i \in N$ and $c_{ij} = 0 \forall (i,j) \in A$, and introduce an additional arc (t,s) with cost $c_{ts} = -1$ and flow bound $u_{ts} = \infty$. Since any flow on arc (t,s) must travel from node s to node t through the arcs in A (since each b(i)=0), the minimum cost flow solution maximizes the flow on arc (t,s).
- Assignment problem, a special class of transportaion problem, here $x_{ij} = 0$ or 1.
- Transportation problem
- Circulation problem, here $b(i) = 0 \forall i \in N$ and we wish to find the circulation with minimum cost.
- Convex cost flow problems, here the cost is a convex function of the amount of flow.
- Generalized flow problems, here arcs may "consume" or "generate" flow, and arcs only conserve flows in the minimum cost flow problem. When x_{ij} units of flow enter an arc (i,j), then $\mu_{ij}x_{ij}$ units arrive at node j, we say the arc is lossy if $0 < \mu_{ij} < 1$ and gainy if $1 < \mu_{ij} < \infty$.
- Multicommodity flow problems

Note that every variant of the network flow problem can be shown to be equivalent to each other:

- Every network flow problem can be reduced to one with exactly one source and exactly one sink node.

- Every network flow problem can be reduced to one without sources or sinks, that is, we can transform the former to a circulation problem.
- Transformation of a node capacity into an arc capacity, just split this node into two nodes with an arc capacity equal to the node capacity.
- The lower bound of arc flow constraint can be reduced to zero, just construct the connection of $\mathbf{y}_{\mathrm{ij}} = \mathbf{x}_{\mathrm{ij}} - \boldsymbol{l}_{\mathrm{ij}}$.
- Inequality constraints $\sum_{\mathrm{j}} \mathbf{x}_{\mathrm{ij}} - \sum_{\mathbf{k}} \mathbf{x}_{\mathbf{ki}} \leq \mathbf{b}_{\mathbf{i}}$: Construct a "dummy node" n+1 and $\mathbf{b}_{n+1} = -B$, where $B = \sum_i b_i$. Any feasible solution for the original problem can be transformed into a feasible solution for the new problem by sending excess flow to node n+1.
- Eliminating upper bounds (Orlin, 2010, Lec. 4): For i with $b(i)$ and j with b(j) and arc with x_{ij}, we transform i with $b(i) - u_{ij}$ and j with $b(j)$ and a new node k with $u(ij)$ and $u_{ij} - x_{ij}$ from k to i and x_{ij} from k to j.

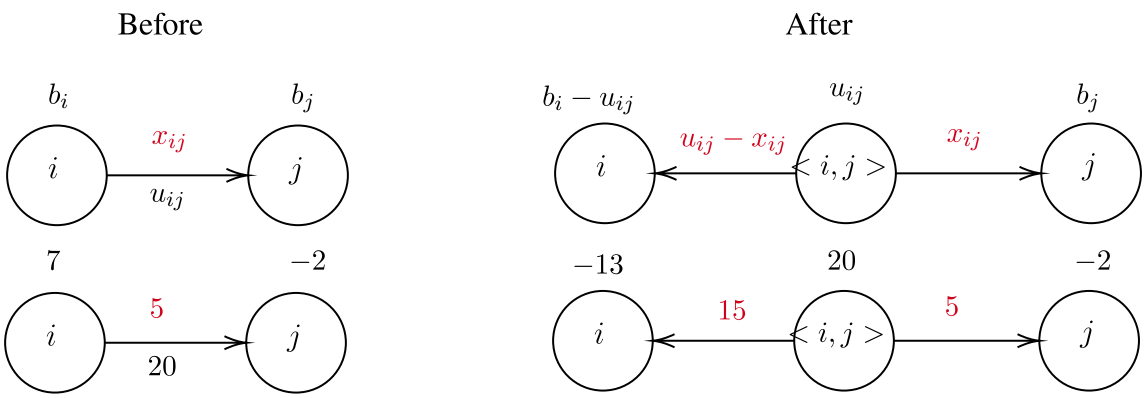

Figure 1: Eliminating upper bounds

- Undirected arcs to directed arcs, this is actually similar to the absolute case in LP. Suppose the arc {i,j} is undirected with cost $c_{ij} \geq 0$ and capacity u_{ij}, we replace each undirected arc by two directed arcs (i,j) and (j,i), both with cost c_{ij} and capacity u_{ij}.
- Arc Reversal (Ahuja et al., 1993, P. 40), this is typically used to remove arcs with negative costs. In this transformation we replace the variable x_{ij} by $u_{ij} - x_{ji}$. Doing so replaces the arc (i,j), which has an associated cost c_{ij}, by the arc (j,i) with an associated cost $-c_{ij}$.

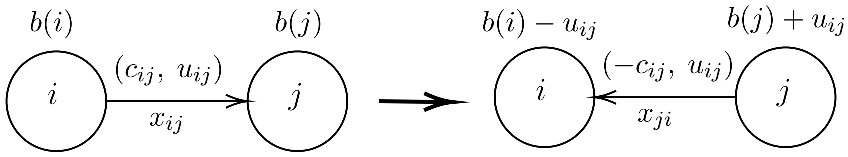

Figure 2: Arc reversal transformation.

There are also two kinds network models does not correlate to flow problems.

- Minimum spanning tree problem
- Matching problem

Definition 2.1 (Circulation)

*Any flow vector **f** that satisfies $Af = 0$ is called a circulation.*

Intuitively, with zero external supply and demand, the flow "circulates" inside the network.

3 Shortest Path Problem

There is some assumptions for this problem,

- All arc lengths are integers. (Can be relaxed)
- The network contains a directed path from node s to every other node in the network.
- The network does not contain a negative cycle.
- The network is directed.

Here we use d(i) denotes the length of some path from s node to node i. And the procedure update(i) means that if $d(j) > d(i) + c_{ij}$ then do $d(j) := d(i) + c_{ij}$ and pred(j):=i, note that distance labels can only decrease in an update step.

Proposition 3.1 (Optimality for subpath)

If the path $s = i_1 - ... - i_k = k$ is a shortest path from node s to node k, then for every $q = 2, ..., k-1$, the subpath $s = i_1 - ... - i_q$ is the shortest path from node s to node i_q.

Proposition 3.2 (Optimality condition 1)

A direct path P from the source node to node k is a shortest path iff $d(j) = d(i) + c_{ij}, \forall (i,j) \in P$, here d(.) denotes the shortest path distance.

Proof If side: Sum up equations $\forall (i,j) \in P$, then you have $d(k) = c_{12} + ...c_{k-1,k}$, and d(.) denotes the shortest path distance, this means that this path is a shortest path.

Onlyif side: ■

Proposition 3.3 (Optimality condition 2 (Malik et al., 1989))

Consider a network without any negative cost cycle. For every node $j \in N$, let $d^s(j)$ denote the length of a shortest path from node s to node j and let $d^t(j)$ denote the length of a shortest path from node j to node t.

- *An arc (i,j) is on a shortest path from node s to node t iff $d^s(t) = d^s(i) + c_{ij} + d^t(j)$.*
- $d^s(t) = min\{d^s(i) + c_{ij} + d^t(j), (i,j) \in A\}$.

There are two kinds of algorithms for solving shortest path problems: label setting and label correcting. The approaches vary in how they update the distance labels from step to step and how they "converge" toward the shortest path distances. Label-setting algorithms designate one label as permanent (optimal) at each iteration. In contrast, label-correcting algorithms consider all labels as temporary until the final step. Label-setting can only apply to acyclic networks and problems with nonnegative arc lengths, while label-correcting are more general and apply to all classes of problems.

3.1 Floyd-Warshall algorithm

Here we use *Floyd-Warshall algorithm* to derive the shortest path, and this is also applicable to the networks with negative arcs

```
Set d(s) = 0 and the remaining distance labels to very large numbers.
Examine the nodes in the topological order and for each order i, scan the arcs
    in A(i).
If d(j) > d(i) + c_ij, for any (i,j) ∈ A(i), then update d(j) = d(i) + c_ij.
After examining all nodes, the distance labels is optimal.
```

3.2 Dijkastra's Algorithm

```
Begin
    S := φ; S̄ := N;
    d(i) := ∞, ∀i ∈ N;
    d(s) := 0, pred(s):=0;
    While |S|<n, do
    Begin
        Let node i ∈ S̄ be such that d(i) = Min{d(j) : j ∈ S̄}
        S := S ∪ {i}; S̄ := S̄ − {i};
        For each (i,j) ∈ A(i), do
            If d(j) > d(i) + c_ij, then d(j) = d(i) + c_ij,
            pred(j):=i;
    End
End
```

Proof [(Borradaile, n.d.)] Reference. ■

This algorithm, also known as label-setting algorithm, maintains two sets of nodes: permanently labeled nodes S and temporarily labeled nodes $\bar{S}$ at each iteration. And the most time-consuming step is at node selection due to distance-label comparison.

Proposition 3.4

The distance labels that the Dijkstra's algorithm designates as permanent are non-decreasing.

Proposition 3.5

If d(i) is the distance label that the algorithm designates as permanent at the beginning of an iteration, then at the end of the iteration, $d(j) \leq d(i) + C$ for each finitely labeled node $j \in \bar{S}$, where C is the maximum arc length.

3.3 Improved Dijkastra's Algorithm

Here we propose some data structures to improve Dijkastra's Algorithm's efficiency. One way is using *Buckets* in Dial's Algorithm.

3.4 Label Correcting Algorithm

Correcting Algorithm is more complicated and can be applied to more general case.

Theorem 3.1 (Optimality Condition)

For every node $j \in N$, let $d(j)$ denote the length of some directed path from the source node to node j. Then, $d(j)$ represents the shortest path distances iff they satisfy the following optimality condition:

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

Actually, this condition can be interpreted as reduced cost condition,, we can define the reduced cost length c_{ij}^d of arc (i, j), where $c_{ij}^d = c_{ij} + d(i) - d(j)$.

Lemma 3.1 (reduced cost property)

- *For any directed cycle W, $\sum_{(i,j)\in W} c_{ij}^d = \sum_{(i,j)\in W} c_{ij}$.*
- *For any directed path P from node k to node l, $\sum_{(i,j)\in P} c_{ij}^d = \sum_{(i,j)\in P} c_{ij} + d(k) - d(l)$.*
- *If $d(.)$ represent shortest path distances, $c_{ij}^d \geq 0$ for every arc $(i, j) \in A$.*

Below is the generic algorithm

```
Begin
    d(s) := 0; pred(s) := 0;
    d(i) := ∞, for i ∈ N − {s};
    While some arc (i,j) satisfies d(j) > d(i) + c_ij, do
    Begin
        d(j) = d(i) + c_ij, pred(j) := i;
    End
End
```

Note on

- *The predecessor indices might not necessarily define a tree. In case of a negative cycle, the resulting list can form a disconnected graph.*
- *We refer to the collection of arcs (pred(j),j) as the predecessor graph, and the label-correcting algorithm satisfies the invariant property that for every arc (i, j) in the predecessor graph, $c_{ij}^d \leq 0$. When the algorithm terminates, the reduced arc length in the predecessor tree must be zero.*

Below is a modified label-correcting algorithm, since the generic algorithm does not specify any method for selecting an arc violating the optimality condition.

```
Begin
    d(s) := 0; pred(s) := 0;
    d(i) := ∞, for i ∈ N − {s};
    LIST:= {S};
    While LIST≠ ∅, do
    Begin
        Remove an element i from LIST
        For each arc (i,j) ∈ A(i), do
            If d(j) > d(i) + c_ij, then
            Begin
                d(j) = d(i) + c_ij; pred(j) := i;
                If j ∉ LIST, then add j to LIST
            End
    End
End
```

3.5 Connection to other topic

3.5.1 Dynamic Lot Sizing

3.5.2 Most vital arc problem (Malik et al., 1989)

3.5.3 Kth shortest path problem

Note that even there are many shortest paths, this algorithm works.

4 Maximum Flow Problem

$$
\begin{aligned}
\max \quad & v \\
\text{s.t.} \quad & \sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = \begin{cases} v & \text{for } i = s \\ 0 & \text{for all } i \in N - \{s \text{ and } t\} \\ -v & \text{for } i = t \end{cases} \qquad (3) \\
& 0 \le x_{ij} \le u_{ij} \quad \text{for each } (i,j) \in A
\end{aligned}
$$

Assumption 4.1

- *The network is directed. (feasibility)*
- *All the capacities are non-negative integers. (feasibility)*
- *The network does not contain a directed path from node s to node t consisting of*

infinite capacity. (bounded, finite optimal)

- *The network does not contain parallel arcs.*

Definition 4.1 (Residual Capacity and Residual Network)

Given a flow x, the residual capacity $r_{ij} = u_{ij} - x_{ij} + x_{ji}$ of arc $(i, j) \in A$ is the maximum additional flow that can be sent from the arcs (i, j) and (j, i) between nodes i and j. Here r_{ij} has two components

- *$u_{ij} - x_{ij}$ is the unused capacity of (i, j).*
- *the current flow x_{ji} on arc (j, i), which can cancel the increase in the flow from i to j.*

We refer to the network $G(x)$ consisting of the arcs with positive residual capacities as the residual network.

By definition, we have $x_{ij} - x_{ji} = u_{ij} - r_{ij}$, since x_{ij} and x_{ji} are positive here, if $u_{ij} \geq r_{ij}$, $x_{ij} = u_{ij} - r_{ij}$ and $x_{ji} = 0$, if $u_{ij} < r_{ij}$, $x_{ji} = r_{ij} - u_{ij}$ and $x_{ij} = 0$.

Definition 4.2 (s-t Cut)

A cut is an $s-t$ cut if $s \in S$ and $t \in \bar{S}$. Capacity of an $s-t$ cut $u[S, \bar{S}] = \sum_{(i,j)\in(S,\bar{S})} u_{ij}$, and this is the upper bound of the flow from s to t. Residual capacity of an $s-t$ cut is $r[S, \bar{S}] = \sum_{(i,j)\in(S,\bar{S})} r_{ij}$.

Let x be a flow in the network. the amount of flow from nodes in S to nodes in $\bar{S}$ can be expressed as follows. Since $0 \leq x_{ij} \leq u_{ij}$, we have $v \leq U[S, \bar{S}]$.

$$v = \sum_{i\in S}\left[\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji}\right] = \sum_{(i,j)\in(S,\bar{S})} x_{ij} - \sum_{(i,j)\in(\bar{S},S)} x_{ij}$$

Lemma 4.1 (Cut's Property)

- *The value of any flow is less than or equal to the capacity of any cut in the network.*
- *For any flow x of value v in a network, the additional flow that can be sent from the source node s to the sink node t is less than or equal to the residual capacity of any $s-t$ cut.*

Any flow x whose value equals the capacity of some cut $[S, \bar{S}]$ is the maximum flow and the cut is the minimum cut. That is, the minimum cut problem is the dual problem of maximum flow problem.

Below is the Generic Augmenting Path Algorithm, Labeling Algorithm and Procedure Augment.

```
Begin
    x := 0;
    while G(x) contains a path from s to t, do
```

```
    Begin
        Identify an augmenting path P from s to t
        δ := Min {r_ij : (i,j) ∈ P}.
        Augment δ units of flow along P and update G(x)
    End
End

Begin
    Label node t;
    While t is labeled, do
    Begin
        Unlabel all the nodes;
        Set pred(j) := 0 for j ∈ N
        Label node s, and LIST:= {s};
        While LIST≠ ∅ and t is unlabeled, do
        Begin
            Remove a node i from LIST;
            For each arc (i,j) in the residual network; do
                If j is unlabeled, set pred(j) := i, label j, add j to LIST
        End
        If t is labeled, then augment.
    End
End

Begin
    Use predecessor labels to trace back from the sink to the source to obtain a
         path P;
    δ := Min {r_ij : (i,j) ∈ P}
    Augment along P
End
```

4.1 Dual: Min-Cut

The dual can be formulated as this way[1] or this way[2], this way[3], this way[4].

A second explanation of Dual[5].

[1] Lecture 15, Stanford University — CS261: Optimization

[2] The dual of the maximum flow problem

[3] Lecture 24: The Max-Flow Min-Cut Theorem Math 482: Linear Programming

[4] Lecture 14: Linear Programming II

[5] Lecture 10: Duality in Linear Programs

Theorem 4.1 (Max-Flow Min-Cut Theorem)

The maximum value of the flow from a source node s to a sink node t in a capacitated network equals the minimum capacity among all $s-t$ cuts.

Theorem 4.2 (Augmenting Path Theorem)

A flow x^* is a maximum flow iff the residual network $G(x^*)$ contains no augmenting path.

Theorem 4.3 (Integrality Theorem)

If all arc capacities are integer, the maximum flow problem has an integer maximum flow.

5 Maximum Flow Problem

$$
\begin{aligned}
\max \quad & v \\
\text{s.t.} \quad & \sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = \begin{cases} v & \text{for } i = s \\ 0 & \text{for all } i \in N - \{s \text{ and } t\} \\ -v & \text{for } i = t \end{cases} \\
& 0 \le x_{ij} \le u_{ij} \qquad \text{for each } (i,j) \in A
\end{aligned} \tag{4}
$$

Assumption 5.1

- *The network is directed. (feasibility)*
- *All the capacities are non-negative integers. (feasibility)*
- *The network does not contain a directed path from node s to node t consisting of infinite capacity. (bounded, finite optimal)*
- *The network does not contain parallel arcs.*

Definition 5.1 (Residual Capacity and Residual Network)

Given a flow x, the residual capacity $r_{ij} = u_{ij} - x_{ij} + x_{ji}$ of arc $(i,j) \in A$ is the maximum additional flow that can be sent from the arcs (i,j) and (j,i) between nodes i and j. Here r_{ij} has two components

- *$u_{ij} - x_{ij}$ is the unused capacity of (i,j).*
- *the current flow x_{ji} on arc (j,i), which can cancel the increase in the flow from i to j.*

We refer to the network $G(x)$ consisting of the arcs with positive residual capacities as the residual network.

By definition, we have $x_{ij} - x_{ji} = u_{ij} - r_{ij}$, since x_{ij} and x_{ji} are positive here, if $u_{ij} \ge r_{ij}$, $x_{ij} = u_{ij} - r_{ij}$ and $x_{ji} = 0$, if $u_{ij} < r_{ij}$, $x_{ji} = r_{ij} - u_{ij}$ and $x_{ij} = 0$.

Definition 5.2 (s-t Cut)

A cut is an $s-t$ cut if $s \in S$ and $t \in \bar{S}$. Capacity of an $s-t$ cut $u[S,\bar{S}] = \sum_{(i,j)\in(S,\bar{S})} u_{ij}$, and this is the upper bound of the flow from s to t. Residual capacity of an $s-t$ cut is $r[S,\bar{S}] = \sum_{(i,j)\in(S,\bar{S})} r_{ij}$.

Let x be a flow in the network. the amount of flow from nodes in S to nodes in $\bar{S}$ can be expressed as follows. Since $0 \leq x_{ij} \leq u_{ij}$, we have $v \leq U[S,\bar{S}]$.

$$v = \sum_{i\in S}\left[\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji}\right] = \sum_{(i,j)\in(S,\bar{S})} x_{ij} - \sum_{(i,j)\in(\bar{S},S)} x_{ij}$$

Lemma 5.1 (Cut's Property)

- *The value of any flow is less than or equal to the capacity of any cut in the network.*
- *For any flow x of value v in a network, the additional flow that can be sent from the source node s to the sink node t is less than or equal to the residual capacity of any $s-t$ cut.*

Any flow x whose value equals the capacity of some cut $[S,\bar{S}]$ is the maximum flow and the cut is the minimum cut. That is, the minimum cut problem is the dual problem of maximum flow problem.

Below is the Generic Augmenting Path Algorithm, Labeling Algorithm and Procedure Augment.

```
Begin
    x := 0;
    while G(x) contains a path from s to t, do
    Begin
        Identify an augmenting path P from s to t
        δ := Min {r_ij : (i,j) ∈ P}.
        Augment δ units of flow along P and update G(x)
    End
End

Begin
    Label node t;
    While t is labeled, do
    Begin
        Unlabel all the nodes;
        Set pred(j) := 0 for j ∈ N
        Label node s, and LIST:= {s};
        While LIST≠ ∅ and t is unlabeled, do
        Begin
            Remove a node i from LIST;
```

```
            For each arc (i,j) in the residual network; do
                If j is unlabeled, set pred(j) := i, label j, add j to LIST
        End
        If t is labeled, then augment.
    End
End

Begin
    Use predecessor labels to trace back from the sink to the source to obtain a
         path P;
    δ := Min {r_ij : (i,j) ∈ P}
    Augment along P
End
```

5.1 Dual: Min-Cut

The dual can be formulated as this way[6] or this way[7], this way[8], this way[9].

A second explanation of Dual[10].

Theorem 5.1 (Max-Flow Min-Cut Theorem)

The maximum value of the flow from a source node s to a sink node t in a capacitated network equals the minimum capacity among all $s-t$ cuts.

Theorem 5.2 (Augmenting Path Theorem)

A flow x^* is a maximum flow iff the residual network $G(x^*)$ contains no augmenting path.

Theorem 5.3 (Integrality Theorem)

If all arc capacities are integer, the maximum flow problem has an integer maximum flow.

6 Network Simplex Algorithm

Definition 6.1 (Free arc and restricted arc)

Arc (i,j) is free if $0 < x_{ij} < u_{ij}$ and is a restricted arc if $x_{ij} = 0$ or $x_{ij} = u_{ij}$.

Definition 6.2 (Cycle-free solution)

A solution x is cycle-free if the network contains no cycle composed only of free arcs.

[6] Lecture 15, Stanford University — CS261: Optimization

[7] The dual of the maximum flow problem

[8] Lecture 24: The Max-Flow Min-Cut Theorem Math 482: Linear Programming

[9] Lecture 14: Linear Programming II

[10] Lecture 10: Duality in Linear Programs

Definition 6.3 (Spanning tree solution)

A feasible solution x and the associated spanning tree of the network is a spanning tree solution if every non-tree arc is a restricted tree. A spanning tree solution partitions the arc set A into three sets (T, L, U):

- *T: $n - 1$ arcs in the spanning tree.*
- *L: the non-tree arcs whose flows are restricted to be zero.*
- *U: the non-tree arcs whose flows are restricted to be the arcs' flow capacities.*

A spanning tree structure is feasible if all arcs' flow satisfy the bounds. The spanning tree is non-degenerate if every tree arc in a spanning tree solution is a free arc.

Lemma 6.1 (Cycle Free Property)

If the objective function of a minimum cost flow problem is bounded from below over the feasible region, the problem always has an optimal cycle free solution.

Lemma 6.2 (Spanning Tree Property)

If the objective function of a minimum cost flow problem is bounded from below over the feasible region, the problem always has an optimal spanning tree solution.

Note on *Similar to simplex method of LP, we can construct a spanning tree solution as a basic solution, e.g., we can set $x_{ij} = 0$ for $(i, j) \in L$, $x_{ij} = u_{ij}$ for $(i, j) \in U$ and solve x_{ij} for $(i, j) \in T$.*

Theorem 6.1 (Optimality Condition)

A spanning tree structure (T, L, U) is an optimal spanning tree structure of the minimum cost flow problem if it is feasible and for some choie of node potential π, the arc reduced costs c^{π}_{ij} satisfy the following conditions:

- $c^{\pi}_{ij} = 0$ *for all* $(i, j) \in \mathbf{T}$.
- $c^{\pi}_{ij} \geq 0$ *for all* $(i, j) \in \mathbf{L}$.
- $c^{\pi}_{ij} \leq 0$ *for all* $(i, j) \in \mathbf{U}$.

Below is the procedure for computing node potentials, where thread(i) is the node in the depth-first traversal search encountered after the node itself.

```
Begin
    π(1) = 0;
    j = thread(1);
    While j ≠ 1, do
        Begin
            i := pred(j);
            If (i,j) ∈ A, then π(j) := π(i) − c_ij;
            If (j,i) ∈ A, then π(j) := π(i) + c_ij;
            j = thread(j).
```

```
    End
End
```

Lemma 6.3 (Dual Integrality Property)

If all arc costs are integer, the minimum cost flow problem always has optimal integer node potentials.

Lemma 6.4 (Primal Integrality Property)

Below is the procedure for computing flows and the Network Simplex Algorithm.

```
Begin
    b^{\prime}(i)=b(i), i \in N;
    For (i,j) ∈ U, do
        set x_ij = u_ij, b'(i) = b'(i) − u_ij, b'(j) = b'(j) + u_ij;
    For (i,j) ∈ L, do
        set x_ij = 0;
    T' := T;
    While T' ≠ {1} do
        Begin
            Select a leaf node j ∈ T';
            i := pred(j);
            If (i,j) ∈ T', then
                x_ij := −b(j);
            Else
                x_ij := b(j).
            b'(i) := b'(i) + b'(j);
            Delete node j and the arc incident to it in T'.
        End
End


Begin
    Determine an initial feasible tree structure (T,L,U);
    Let x be the flow and π the node potentials associated with tree;
    While some non-tree arcs violate optimality condition, do
        Begin
            Select an entering arc (k,l) violating the optimality condition;
            Add (k,l) to the tree and determine the leaving (p,q);
            Perform a tree update, update the flow x and node potential π.
        End
End
```

Note on Entering variable *Choosing* $(i,j) \in L$*, with* $c^{\pi}_{ij} < 0$ *or* $(i,j) \in U$*, with* $c^{\pi}_{ij} > 0$*. The* *standard for selecting can be either the largest* $\left|c_{ij}^{\pi}\right|$ *or the first arc scanned.*

Note on Pivoting *Suppose we choose* (k,l) *as entering variable, and after that we get the cycle* w*, which is also called as pivot cycle.*

- *Let the orientation of the cycle* W *be that of* (k,l) *if* $(k,l) \in L$ *or the opposite to that of* (k,l) *if* $(k,l) \in U$.
- $\bar{W}$ *and* $\underline{W}$ *are respectively the forward and backward arc sets.*
- *The maximum flow change* δ_{ij} *satisfies that* $\delta_{ij} = u_{ij} - x_{ij}$ *if* $(i,j) \in \bar{W}$*, and* $\delta_{ij} = x_{ij}$ *if* $(i,j) \in \underline{W}$.
- *Augment* $\delta = \mathrm{Min}\,\{\delta_{ij} : (i,j) \in W\}$*, and the arc that defines* δ *leaves the basis.*

7 Lagrangian Relaxation

If LP's constraints can be divided into two types: some are easy to solve, and the others are not easy to solve, than we can use Lagrangian relaxation to remove "bad" constraints and putting them into the objective function, assigned with weights (the Lagrangian multiplier).

7.1 Symmetric Form

	Primal		Lagrangian Relaxation	Lagrangian multiplier problem
min	$c^T x$	min	$cx + \mu(Ax - b)$	$L^* = \max\limits_{\mu} L(\mu)$
s.t.	$Ax = b$	s.t.	$x \in X$	$L(\mu) = \min\{cx + \mu(Ax - b) : x \in X\}$
	$x \in X$ a polyhedral set.			

(5)

Theorem 7.1 (Lagrangian Bounding Pricinple)

For any vector μ *of the Lagrangian multipliers, the value* $L(\mu)$ *of the Lagrangian function is a lower bound on the optimal objective function value* z^* *of the original optimization problem.*

Theorem 7.2 (Weak Duality)

The optimal objective function value L^* *of the Lagrangian multiplier problem is always a lower bound on the optimal objective function value of the original problem (i.e.,* $L^* \le z^*$*).*

Theorem 7.3 (Optimality Test)

- *Suppose that* μ *is a vecotr of Lagrangian multipliers and* x *is a feasible solution to the Primal problem satisfying the condition* $L(\mu) = cx$*. Then* $L(\mu)$ *is an optimal solution of the Lagrangian multiplier problem (i.e.* $L^* = L(\mu)$*) and* x *is an optimal solution to the Primal problem.*
- *If for some choice of the Lagrangian multiplier vector* μ*, the solution* x^* *of the*

Lagrangian relaxation is feasible in the Primal problem, then x^* is an optimal solution to the Primal problem and μ is an optimal solution to the Lagrangian multiplier problem.

7.2 Asymmetric Form

	Primal		**Lagrangian Dual**
min	$z(x) = c^T x$	max	$f(w) = w^T b + \min_{x \in X}(c^T - w^T A)x$
s.t.	$Ax \geq b$	s.t.	$w \geq 0$
	$x \in X$, where X is a polyhedral set.		

(6)

Theorem 7.4 (Weak Duality)

The optimal objective function value L^* of the Lagrangian multiplier problem is always a lower bound on the optimal objective function value of the original problem (i.e., $f(w^*) \leq z(x^*)$).

Proof This is equal to show any feasible solution x_0 to Primal and any feasible solution w_0 to Lagrangian Dual satisfy $c^T x_0 \geq f(w_0)$. Since x_0 is feasible to Primal, $x_0 \in X$ and $\min_{x \in X}(c^T - w^T A)x \leq (c^T - w^T A)x_0$. Note that $Ax_0 \geq b$ means $w_0^T A x_0 \geq w_0^T b \quad (w_0 \geq 0)$, thus

$$f(w_0) \leq w_0^T b + (c^T - w^T A)x_0 = c^T x_0 + w_0^T b - w^T A x_0 \leq c^T x_0$$

■

Theorem 7.5 (Strong Duality)

Suppose that X is nonempty and bounded and that the primal problem possess a finite optimal solution. Then

$$\min_{Ax \geq b, x \in X} c^T x = \max_{w \geq 0} f(w)$$

Proof

$$\begin{array}{llll} & \textbf{Primal} & & \textbf{Dual} \\ \min & z(x) = c^T x & \max & \lambda_1^T b + \lambda_2^T d \\ \text{s.t.} & Ax \geq b & \text{s.t.} & \lambda_1^T A + \lambda_2^T B = c^T \\ & Bx \geq d & & \lambda_1, \lambda_2 \geq 0 \end{array} \tag{7}$$

Note that $x \in X$ can be expressed as $Bx \geq d$, and assume x^* is a feasible optimal solution to Primal, λ_1^* and λ_2^* are dual vector for constraint $Ax \geq b$ and $Bx \geq d$, then we must have dual feasiblity

$$(\lambda_1^*)^T A + (\lambda_2^*)^T B = c^T \tag{8}$$

and following complementary slackness conditions

$$\begin{cases} \lambda_1^*(Ax-b)=0 \\ \lambda_2^*(Bx-d)=0 \\ x^*((\lambda_1^*)^T A+(\lambda_2^*)^T B-c^T)=0 \end{cases}$$

Since $\lambda_1^* \geq 0$, λ_1^* is also a feasible solution to $\max_{w\geq 0} f(w)$.

$$f(\lambda_1^*)=(\lambda_1^*)^T b+\min_{x\in X}(c^T-(\lambda_1^*)^T A)x$$

Consider the following duality, note that x^* and λ_2^* are optimal solution to Primal and Dual respectively because of primal feasiblity, dual feasibility (8) and complementary slackness conditions (7.2, 7.2).

$$\begin{array}{llll} & \textbf{Primal} & & \textbf{Dual} \\ \min & z(x)=(c^T-(\lambda_1^*)^T A)x & \max & \lambda_2^T d \\ \text{s.t.} & Bx\geq d & \text{s.t.} & \lambda_2^T B=c^T-(\lambda_1^*)^T A, \lambda_2\geq 0 \end{array} \tag{9}$$

Thus $f(\lambda^*)=c^T x^*$, and by weak duality theorem we know $f(w)\leq c^T x^*=f(\lambda^*)=c^T x$, thus λ^* is also the optimal solution to the Lagrangian dual and the optimal solutions for both questions are equal. ■

Bibliography

Ahuja, Ravindra, Thomas Magnanti, and James Orlin (Feb. 1993). *Network Flows: Theory, Algorithms, and Applications*. 1st edition. Englewood Cliffs, N.J: Pearson. ISBN: 978-0-13-617549-0.

Borradaile, Glencora (n.d.). *Dijkstra's Algorithm: Correctness by Induction*. Oregon State University.

Malik, K., A. K. Mittal, and S. K. Gupta (Aug. 1989). "The k Most Vital Arcs in the Shortest Path Problem". In: *Operations Research Letters* 8.4, pp. 223–227.

Orlin, James (2010). *15.082J/6.855J/ESD.78J Network Optimization*. Massachusetts Institute of Technology.