Chapter 6: Network Flow Problems

(Bertsimas & Tsitsiklis, Chapter 7)
Min-Cost-Network-Flow (Sec. 7.2 in [BT])

Def.: A network \( G = (N, A) \) is a directed acyclic graph together with additional information:

- \( b_i, i \in N \) is the supply at each node (sometimes the negative: “demands” are provided)
  - Each node with \( b_i > 0 \) is called a source
  - Each node with \( b_i < 0 \) is called a sink
- \( u_{ij} \geq 0 \), possibly infinite are the capacities
- \( c_{ij} \in \mathbb{R} \) is the cost per unit flow along the arc \((i, j)\)

\[
 b_i + \sum_{j \in \delta^+(i)} f_{j,i} = \sum_{j \in \delta^-(i)} f_{i,j} \quad \forall i \in N \\
 0 \leq f_{i,j} \leq u_{i,j} \quad \forall (i, j) \in A
\]

A vector \( f \) satisfying these is called a flow.

The Minimum Cost Network Flow problem asks to minimize

\[
 \sum_{(i,j) \in A} c_{i,j} f_{i,j}
\]

with respect to the flow constraints.

If all \( u_{i,j} = \infty \), the problem is called uncapacitated, otherwise capacitated.
Special cases of Min-Cost-Network-Flow

Special cases include:

- **max s-t-flow**: connect $t$ to $s$ via an uncapacitated edge of cost $-1$. No sources and no sinks required.

- The transportation problem: Bipartite graph: Vertices are partitioned into left and right vertices. Only arcs from left to right. Every min-cost-network-flow problem can be written as a transportation problem ([BT, Exerc. 7.6])

- The assignment problem: Special case of the transportation problem: Same number of left and right vertices, all demands and supplies are 1. We will see: An optimum can always be found such that all flow values are from $\{0, 1\}$. This solves the max-weight matching problem (slide 121)!
Min-cost-network-flow problems without sources and sinks (circulation problems)

Theorem [BT, p. 274,275]

For every min-cost-network-flow problem there exists an equivalent min-cost-network-flow problem without any sources or sinks.

Proof: Two-step process: First reduce to a single source and single sink. Then remove these. More precisely ...
Simplex Algorithm for the Min-Cost-Network-Flow problem

We introduce next the building blocks of the uncapacitated network simplex algorithm:

- Basic solutions $\leftrightarrow$ tree solutions
- Basic directions $\leftrightarrow$ simple circulations (flow around a cycle)
- Reduced costs $\leftrightarrow$ cost of a cycle

We know these objects, but they are particularly easy for the uncapacitated Min-Cost-Network-Flow problem.
Circulations and cycles (end of Sec. 7.2 in [BT])

**Definition:** A flow vector $f$ that satisfies $Af = 0$ is called a circulation.

Intuition: Flow conservation, but zero supply and demand. The flow “circulates” within the network.

**Definition:** An undirected walk on a network $G = (N, A)$ from node $i_0$ to node $i_t$ is a finite sequence of nodes such that for each $k$, $1 \leq k \leq t$, either $(i_{k-1}, i_k) \in A$ or $(i_k, i_{k-1}) \in A$. An undirected walk with $i_0 = i_t$ and where all $i_k$, $1 \leq k \leq t$ are pairwise distinct is called a cycle.

**Definition:** Let $C$ be a cycle and let $F$ be the set of forward arcs and $B$ be the set of backwards arcs in $C$. The flow vector $h^C$ with components

$$h^C_{ij} = 1 \text{ if } (i, j) \in F; \quad h^C_{ij} = -1 \text{ if } (i, j) \in B; \quad h^C_{ij} = 0 \text{ otherwise}$$

is called the simple circulation associated with $C$.

The cost of the cycle $C$: $c^T h^C = \sum_{(i,j) \in F} c_{i,j} - \sum_{(i,j) \in B} c_{i,j}$

- $Ah^C = 0$, thus $h^C$ is a circulation.
- If $f$ is a flow, and $C$ is a cycle, and $\theta \in \mathbb{R}$, then we say that the flow $f + \theta h^C$ is obtained from $f$ by pushing $\theta$ units of flow around $C$.
- One can think of $f$ as a point and $h^C$ as a direction vector.
A cycle in an undirected graph is defined analogously to the directed case.

A connected graph without cycles is called a tree.

A subgraph of $G$ containing all nodes in $V$ is called spanning.

**Theorem 7.2 in [BT]**

Let $G$ be connected. Given any subset $E_0$ of the set of edges that does not contain a cycle. Then $E_0$ can be augmented with additional edges to form a spanning tree.

Proof:

- If $G$ is a tree, choose all edges to form the spanning tree.
- If $G$ is not a tree, there must be a cycle in $G$. By assumption the cycle contains an edge that is not in $E_0$. Remove that edge (the resulting graph remains connected) and proceed by induction.
The network simplex: Linearly independent rows

Observation: Adding up all rows in $A$ gives the zero vector, hence the rows are linearly dependent!

Solution: Remove the last row of $A$ to obtain $\tilde{A}$. We will see that the rows of $\tilde{A}$ are linearly independent. Also remove the last row of $b$ to obtain $\tilde{b}$.

Definition 7.1 in [BT]

A flow vector $f$ is called a tree solution if it can be constructed by the following procedure:

- Pick a set $T \subseteq A$ of arcs that form a spanning tree (ignoring directions of the arcs)
- Let $f_{i,j} = 0$ for all $(i,j) \notin T$
- Use the flow conservation equation $\tilde{A}f = \tilde{b}$ to determine the flow values $f_{i,j}$ for $(i,j) \in T$.

A tree solution that also satisfies $f \geq 0$ is called a feasible tree solution.

Example ...

Once a tree is fixed, the tree solution is uniquely determined (Theorem 7.3).

Easy to see, but the formal proof is illuminating:

Corollary 7.1 in [BT]

$\tilde{A}$ has linearly independent rows.
The network simplex: Basic solutions are tree solutions

We have just seen: A tree solution is a basic solution.

Theorem 7.4 in [BT]

Every basic solution is a tree solution.

Proof ...
Change of basis ([BT] p. 284): basic directions and reduced costs

\((k, l)^{th}\) basic direction
- Given a feasible tree solution \(f\) with tree \(T\). Take a nonbasic variable \(f_{k,l}\). Goal: Increase \(f_{k,l}\) while preserving \(Af = b\).
- \(T\) together with the arc \((k, l)\) contains a unique cycle \(C\) the traverses \((k, l)\) in forward direction.
- The \((k, l)^{th}\) basic direction is \(d = h^C\).
- Intuitive we push flow around the cycle \(C\) (which increases the flow on the arc 
\((k, l)\)) until the flow value on an arc becomes zero.
- The arc whose flow value becomes zero leaves the basis.

Formally, we go from \(f\) to \(f + \theta^* h^C\).

\((k, l)^{th}\) reduced costs
- The \((k, l)^{th}\) reduced costs \(\overline{c}_{(k,l)}\) is the cost change if do one unit into the \((k, l)^{th}\) basic direction
- This is the sum of costs around the cycle \(C\) (backwards arcs count negatively):

Let \(F\) be the set of forward arcs in \(C\) and \(B\) be the set of backwards arcs in \(C\). Then

\[
\overline{c}_{(k,l)} = \sum_{a \in F} c_a - \sum_{b \in B} c_b
\]
Anticycling and finding an initial solution

- **Anticycling** can be achieved for example with Bland’s rule.

**Slight caveat:** The variables have two indices. Define a lexicographic ordering:

\[(i, j) < (k, l) \text{ iff } i < k \text{ or } (i = k \text{ and } j < l)\]

This gives a total order: We always have \((i, j) < (k, l)\) or \((k, l) < (i, j)\) or \((i, j) = (k, l)\).

- **Finding an initial solution** (uncapacitated):
  - For each pair of source-sink nodes \((s, t)\) we introduce an uncapacitated arc from \(s\) to \(t\) with large cost \(M\).
  - We find a feasible tree solution by selecting any undirected tree and sending flow from sources to sinks over the corresponding directed arcs. After augmenting the set of used arcs to a tree, this results in a feasible tree solution.
  - Since \(M\) is large, no optimum uses the auxiliary arcs (if a feasible flow of the original problem exists).
Complete description of (non-optimized) network-simplex algorithm

1. Construct auxiliary arcs with large cost $M$ and an initial feasible tree solution $f$ with tree $T$.

2. Compute reduced costs (sum of costs around the cycle) for all arcs not in $T$.

3. If all reduced costs $\geq 0$: Terminate. Otherwise: Choose arc $(k, l)$ with $\bar{c}_{(k,l)} < 0$. If several, take the one with smallest (lexicographically) index pair (Bland’s rule).

4. Let $C$ be the cycle formed by $(k, l)$ and arcs in $T$.

5. Let $\theta^*$ be the max flow value of the backwards arcs in $C$. These are the blocking arcs.

6. Update $f$ to $f + \theta^* h^C$. The arc $(k, l)$ enters the basis. The blocking arc with the smallest lexicographical index leaves the basis (Bland’s rule).

7. Continue with step 2.
# Integrality of the optimal solution

<table>
<thead>
<tr>
<th>Theorem 7.5 in [BT]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consider an uncapacitated min-cost-network-flow problem and assume that the underlying graph is connected.</td>
</tr>
<tr>
<td>(a) For every basis matrix $B$, the matrix $B^{-1}$ has integer entries.</td>
</tr>
<tr>
<td>(b) If the supplies $b_i$ are integer, then every basic solution has integer coordinates (i.e., every tree solution has integer flow values).</td>
</tr>
<tr>
<td>(c) If the cost coefficients $c_{i,j}$ are integer, then every basic solution to the dual problem has integer coordinates.</td>
</tr>
</tbody>
</table>

**Proof...**

<table>
<thead>
<tr>
<th>Corollary 7.2 in [BT]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consider an uncapacitated min-cost-network-flow problem and assume that the opt. cost is finite.</td>
</tr>
<tr>
<td>(a) If all supplies $b_i$ are integer, then there exists an integer optimal flow.</td>
</tr>
<tr>
<td>(b) If all cost coefficients $c_i$ are integer, then there exists an integer optimal solution to the dual problem.</td>
</tr>
</tbody>
</table>

This means that max-weight-matching problem (which is a combinatorial problem) can be solved with an LP!
The Negative-Cost-Cycle Algorithm ([BT] p. 294)

We discussed the uncapacitated network simplex algorithm. The algorithm with capacities is similar, with similar properties.

We want to discuss another similar and important optimisation algorithm for the capacitated min-cost-network-flow problem: The negative cost cycle algorithm.

**Definition [BT] p.294**

Given a flow $f$ in a capacitated network, a cycle $C$ is called **saturated** if $f + \theta h^C$ is infeasible for all $\theta > 0$. Otherwise $C$ is called **unsaturated**.

**Observation:**

- An unsaturated cycle gives a feasible direction.
- A saturated cycle gives an infeasible direction.
- Of course, there can be feasible directions $d$ that do not come from cycles, i.e., $d \neq h^C$ for all cycles $C$.

Define

$$\theta^*(C) := \min \left\{ \min_{a \in F} (u_a - f_a), \min_{a \in B} f_a \right\}$$

This is the largest $\theta$ such that $f + \theta(C)h^C$ is feasible.

$C$ is saturated if $\theta^*(C) = 0$. $C$ is unsaturated if $\theta^*(C) > 0$. 
The Negative-Cost-Cycle Algorithm

1. Start with a feasible flow $f$.
2. Search for an unsaturated cycle with negative cost.
3. If no unsaturated cycle with negative cost can be found: terminate.
4. If an unsaturated cycle $C$ with negative cost is found:
   a. If $\theta^*(C) < \infty$, update $f$ to $f + \theta^*(C)h^C$ and go to (2).
   b. If $\theta^*(C) = \infty$, terminate (the optimal cost is $-\infty$).

▶ Observation: The objective function value decreases in each step (in contrast to the simplex algorithm).

Remaining questions:
(a) How do we start the algorithm?
(b) How do we search for an unsaturated cycle with negative cost?
(c) If the algorithm terminates, is the solution optimal?
(d) Is the algorithm guaranteed to terminate?

We answer them on the upcoming slides.
Starting the algorithm

- We are in a setting with arc capacities! Hence:

- Remove all sources and sinks.

- Now the zero flow is a feasible solution.
Termination

- If all arc capacities are integers or infinite, and if we start the algorithm with an integer flow, then the flow remains integer in every iteration.
- Remark: If the algorithm terminates, then the solution is either an integer flow, or the optimal cost is \(-\infty\).

**Theorem 7.7 in [BT]**

If all arc capacities are integers or infinite, and if we start the algorithm with an integer flow, and if the optimal cost is finite, then the algorithm terminates after a finite number of steps.

Proof: In each iteration the cost decreases by at least \(v\), where \(v\) is the minimum cost value of all negative cycles in the network.

Caveat 1: If the capacities are not integers, the algorithm may make smaller and smaller improvements in every step and never terminate.

Caveat 2: If the optimal cost is \(-\infty\), the algorithm may not detect that in finite time.
Find an unsaturated cycle with negative cost: The residual network

**Definition [BT] p. 295**

For a capacitated network $G$ and a feasible flow $f$ define the residual network $R(G, f)$ as follows:

- The node set of $R(G, f)$ is the same as the node set of $G$.
- For each arc $(i,j)$ in $G$ we have two arcs $(i,j)$ and $(j,i)$ in $R(G, f)$. Costs and capacities are depicted below. **Delete all arcs with zero capacity.**

- A feasible circulation $d'$ on $R(G, f)$ gives a flow $d$ on $G$ with $f + d$ feasible.
- Every flow $d$ on $G$ with $f + d$ feasible gives rise to a feasible circulation $d'$ in $R(G, f)$.
- The costs of $d$ and $d'$ are the same!
- Corollary: Every unsaturated cycle in $G$ corresponds to a directed cycle in $R(G, f)$ and vice versa!
Negative cost cycles: Bellman-Ford

Task: To find directed cycles of negative cost in a directed graph with cost values.
Key concept: The cost of a path is the sum of the costs of its arcs.

Bellman-Ford algorithm ([BT] p. 336):

- Fix a node $s$.
- At iteration $i$ the Bellman-Ford algorithm computes the costs $d_i(v)$ of the cheapest path from $s$ to $v$ that uses at most $i$ arcs. It also stores a predecessor list $pr$ so that the cheapest cycle can be reconstructed after the run.

- Initialize $d_0(s) = 0$ and $d_0(v) = \infty$ for all $v \neq s$.
- Step $i \to i + 1$: For each $v$ set
  \[ d_{i+1}(v) := \min\{d_i(v), \min_{arc \ (w, v)}\{d_i(w) + c_{w,v}\}\}. \]
  - If an arc $(w, v)$ was chosen as the minimum (instead of $d_i(v)$), then $pr(v) := w$.
  - If $d_i(v)$ was chosen as the minimumum, then keep $pr(v)$ unchanged.
- If $d_{i+1}(s) < 0$, then output
  “Negative cost cycle found. Predecessors stored in $pr$”

Correctness can be proved by induction. If Bellman-Ford finds $d_i(s) < 0$, then a negative cost cycle is found. If $s$ is contained in a negative cost cycle with $k$ arcs, then Bellman-Ford finds it after $k$ iterations. We run Bellman-Ford for every node $s$ for at most $|V|$ iterations. This ensures that we find a negative cost cycle if one exists.
Bellman-Ford Example
Optimality conditions

**Theorem 7.6 in [BT]**

A feasible flow \( f \) is optimal if and only if there is no unsaturated cycle with negative cost.

Proof: ...

**Lemma 7.1 in [BT] “Flow Decomposition”**

Let \( f \geq 0 \) be a 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, the \( a_i \) can be chosen to be integers.