# Review session: Exam question 3
# Network Flows and
# Minimum Cost Flow algorithms.

- Lecture 9
  - Totally Unimodular Linear Programs. Network Flows: Minimum Cost Flows.
  - Course notes: "Network Flows", Sections 1-4.
- Lecture 10
  - Network Flows: Maximum $(s,t)$ –flows.
  - Course notes: "Network Flows", Sections 5-6.
- Lecture 11
  - The primal and dual network simplex algorithm.
  - Vanderbei, Chapter 14 (pp 199-213).
- Lecture 12
  - The cycle cancelling algorithm.
  - Course notes: "Network Flows", Sections 7.

# Totally Unimodular Matrices

**Definition**

An $m \times n$ matrix $A = (a_{ij})$ is *totally unimodular* if every square submatrix has determinant in $\{-1,0,1\}$.

**TUM Integrality Theorem**

Let $A = (a_{ij})$ be $m \times n$ totally unimodular matrix and let $b \in \boldsymbol{Z}^m$.

Then all basic solutions of

$$F = \{Ax \leq b, x \geq 0\}$$

are integer.

# Cramer's Rule

The system $Ax = b$ has a unique solution if and only if $\det(A) \neq 0$, and in that case it is given by

$$x_i = \frac{\det(A_i)}{\det(A)}$$

where $A_i$ is $A$ with column $i$ replaced by $b$.

Alternatively recall:

$$A^{-1} = \frac{\text{adj}(A)}{\det(A)}$$

# Dictionaries in Matrix form

$$Ax = [B \quad N]\begin{bmatrix} x_{\mathcal{B}} \\ x_{\mathcal{N}} \end{bmatrix} = Bx_{\mathcal{B}} + Nx_{\mathcal{N}} = b$$
$$\zeta = c^{\top}x = [c_{\mathcal{B}}^{\top} \quad c_{\mathcal{N}}^{\top}]\begin{bmatrix} x_{\mathcal{B}} \\ x_{\mathcal{N}} \end{bmatrix} = c_{\mathcal{B}}^{\top}x_{\mathcal{B}} + c_{\mathcal{N}}^{\top}x_{\mathcal{N}}$$

Primal dictionary:

$$\zeta = c_{\mathcal{B}}^{\top}B^{-1}b - \left((B^{-1}N)^{\top}c_{\mathcal{B}} - c_{\mathcal{N}}\right)^{\top}x_{\mathcal{N}}$$
$$x_{\mathcal{B}} = B^{-1}b - B^{-1}Nx_{\mathcal{N}}$$

- TUM implies that $\det(B) = \pm 1$, and thus $B^{-1}$ is an integer matrix.

# Important example:
# The node-arc adjacency matrix

Let $D = (N, A)$ be directed graph and $A$ its node-arc adjacency matrix.

- Rows are indexed by nodes. Columns by arcs.

Entry $(k, (i, j))$ is:

- $1$ when $k = i$
- $-1$ when $k = j$
- $0$ otherwise.

**Lemma 1**

If $A = (a_{ij})$ is a matrix with entries from $\{-1,0,1\}$ such that

every column of $A$ has at most one entry that is 1 and at most one entry that is $-1$, then $A$ is totally unimodular.

# Constructing further TUM matrices

**Lemma 2**

TUM is preserved by

- Transposing matrix.
- Removing a row or a column.
- Multiplying a row or a column by $-1$.

**Lemma 3**

If $A$ is TUM then $[A \quad I]$ and $[A \quad A]$ are TUM as well.

# Flow Networks

Directed graph $D = (N, A)$.

Flow $x$ assigns a real number $x_{ij}$ to arc $ij \in A$.

Nonnegativity constraint: $x_{ij} \geq 0$

- Outgoing flow from node $i$: $\sum_{ij \in A} x_{ij}$
- Ingoing flow from node $i$: $\sum_{ji \in A} x_{ij}$
- Balance at node $i$ w.r.t. $x$:

$$b_i(x) = \sum_{ij \in A} x_{ij} - \sum_{ji \in A} x_{ij}$$

- Note $i$ is *source* if $b_i(x) > 0$, and *sink* if $b_i(x) < 0$.
- No sources or sinks: flow is *circulation*.

# Constraints

- Balance constraints (required):
  - $b_i(x) = b_i$ for all $i \in N$.
  - Assumption: $\sum_{i \in N} b_i = 0$
- Arc constraints (optional):
  - $l_{ij} \leq x_{ij} \leq u_{ij}$ for all $ij \in A$.
  - Assumption: $0 \leq l_{ij} \leq u_{ij}$

# The minimum cost problem

Given network $D = (N, A)$ with *arc costs* $c_{ij}$, together with balance constraints and possibly arc constraints.

Find feasible flow $x$ minimizing

$$\sum_{ij \in A} c_{ij} x_{ij}$$

**Integrality theorem**

If all balance constraints, lower bounds, and upper bounds are integer, then there is a minimum cost feasible flow that is integer.

**Proof:** By TUM of LP.

Examples: Transportation problem, Tanker Scheduling Problem, Optimal loading of a hopping airplane, …

# Maximum $(s, t)$-flow problem

Given network $D = (N, A)$ with upper bounds $u$ and specified source $s \in N$ and sink $t \in N$.

Feasibility criterion:

- Non-negativity: $x_{ij} \geq 0$
- Flow conservation: $b_i(x) = 0, \quad$ for $i \neq s, t$.
- Arc upper bounds respected: $x_{ij} \leq u_{ij}$

Find feasible flow $x$ maximizing the *value* $|x| = b_s(x)$.

Can be seen as special case of minimum cost flow problem: Add $(t, s)$ arc with cost $-1$, forming a circulation network.

# Max-flow Min-cut Theorem

$(s,t)$-cut: Partition $(S,T)$ of nodes $N = S \cup T$ with $s \in S$ and $t \in T$.

Minimum $(s,t)$-cut problem: Find $(s,t)$-cut $(S,T)$ minimizing the *capacity* $u(S,T) = \sum_{i \in S, j \in T} u_{ij}$.

Let $x$ be a feasible $(s,t)$-flow and let $(S,T)$ be $(s,t)$-cut.

Then: $|x| \leq u(S,T)$ (compare to weak duality)

**Theorem**

Let $x$ be a maximal value feasible (s,t)-flow and let $(S,T)$ be a minimum capacity (s,t)-cut. Then: $|x| = u(S,T)$ (compare to strong duality)

**Proof:** By strong duality theorem for LP.

# Examples

- Maximum $(s, t)$-flow:
  - Preemptive scheduling on uniform parallel machines.
  - Distributed computing on two-processor systems.
  - …

- Minimum $(s, t)$-cut:
  - Open pit mining.
  - …

# Algorithms for minimum cost flow

- Network simplex algorithm
  - Specialization of the simplex algorithm to flow networks.
  - Comes in both primal and dual variant.
  - Very hard to beat in practice.

- Klein's cycle cancelling algorithm
  - Generalization of the Ford-Fulkerson max flow algorithm.
  - A variation where augmenting cycles are chosen to be of *minimum mean cost* is a *strongly polynomial time* algorithm (like the Edmonds-Karp max flow algorithm).
  - Not really competitive in practice.

- Many other types of algorithms known.
  - So-called cost-scaling algorithms (which are  polynomial time algorithms) can sometimes be competitive with network simplex on very large networks. (Compare to the case of interior point algorithms for LP).

# LP formulation and the dual

**Primal:**

Minimize $\sum_{ij \in A} c_{ij} x_{ij}$

Subject to $\sum_{ji \in A} x_{ji} - \sum_{ij \in A} x_{ji} = -b_i$

$x_{ij} \geq 0$

**Dual:**

Maximize $-\sum_{i \in N} b_i y_i$

Subject to $y_j - y_i \leq c_{ij} \qquad ij \in A$

Not in standard form. Simplex algorithm could still be executed if constraint matrix is of full row rank. If network is connected we would only need just remove one row.

# Complementary Slackness

**Primal:**

Minimize $\sum_{ij \in A} c_{ij} x_{ij}$

Subject to $\sum_{ji \in A} x_{ji} - \sum_{ij \in A} x_{ji} = -b_i$

$x_{ij} \geq 0$

**Dual:**

Maximize $-\sum_{i \in N} b_i y_i$

Subject to $y_j - y_i + z_{ij} = c_{ij} \qquad ij \in A$

$z_{ij} \geq 0$

Feasible solutions $x$ and $(y, z)$ are (both) optimal if and only if $x_{ij} z_{ij} = 0$ for all $ij \in A$ .

# Basis = Spanning Tree

- A basic solution corresponds to a spanning tree T of the network.
- Fix a root $r$ of T.
- Erasing the row of the node-arc incidence matrix $A$ corresponding to $r$ and erasing columns corresponding to arcs not in T results in a non-singular matrix (the basis matrix $B$).

Given T we can compute basic primal and dual solutions:

- Primal solution:
  - Assign flow 0 to edges not in T (non-basic variables)
  - Compute flow on edges in T (basic variables) working inward from leaves.
- Dual solution:
  - Compute dual variables (potentials), setting $y_r = 0$, and solving $y_j = y_i + c_{ij}$ for all $ij \in T$, also letting $z_{ij} = 0$.
  - Compute remaining dual slacks $z_{ij} = c_{ij} - y_j + y_i$, for $ij \notin T$.

# Understanding dual solution

- The dual variable $y_i$ can also be computed by:
  - Consider path from $r$ to $i$. Add costs of forward edges and subtract costs of backward edges.
- The dual slack $z_{ij}$ for $ij \notin A$ can also be computed by:
  - Consider cycle formed by $i$ and T in direction of $ij$. Add costs of forward edges and subtract costs of backward edges.

# Primal and Dual Network Simplex

- Primal algorithm
  - Select *entering arc*: Pick some $ij \in A$ with $z_{ij} < 0$.
  - Select leaving arc: Consider cycle formed by $ij$ and T in the direction of $ij$. Pick backwards edge with minimum flow value. Send flow along cycle to cancel flow on this edge.
  - Update dual solution

- Dual algorithm
  - Select leaving arc: Pick an arc $ij \in T$ with $x_{ij} < 0$.
  - Select entering arc: Find an arc that with T with forms cycle in direction of $ij$. Pick such arc with minimum dual slack. Send flow along cycle to increase flow on arc $ij$ to 0.
  - Update dual solution

# Initialization, Termination, Complexity,…

Similar issues as the general Simplex algorithm.

- Initialization:
  - Two-phase algorithm. Either construct auxiliary program (auxiliary network) or use dual based phase 1.
- Termination:
  - Cycling remains an issue that must be handled by anti-pivoting rules.
  - In network simplex there are in fact specialized pivot rules that avoid *stalling*, long sequences of pivot operations that do not improve the solutions.
- Complexity
  - Worst-case exponential.
  - Very efficient in practice.

# Klein's cycle cancelling algorithm

- Like the (primal) network simplex algorithm:
  - Maintain a feasible flow $x$.
  - Improve the cost of $x$ by changing flow along a cycle of the network (increasing flow on *forward arcs* and decreasing flow on *backward arcs*).
- Unlike the network simplex algorithm:
  - Do not restrict to *tree flows* (i.e. basic solutions).
  - Consider *any* possible cycle in the network.
- We will describe the algorithm for networks *with* arc constraints.

# Cycle flows and augmenting cycles

Let $C$ be a cycle in $D$. Let $F$ be set of forward arcs and B be set of backward arcs.

- Cost of C: $c(C) = \sum_{ij \in F} c_{ij} - \sum_{ij \in B} c_{ij}$
- Cycle flow $\gamma_C^{\delta}$:
  - On arcs in $F$: $\delta$
  - On arcs in $B$: $-\delta$
  - Other arcs : 0

C is an *augmenting cycle* if

1. Cost is negative: $c(C) < 0$.
2. There exists $\delta > 0$ such that $x + \gamma_C^{\delta}$ is feasible.

$$\delta(C) = \min\left\{\min_{ij \in F}(u_{ij} - x_{ij}), \min_{ij \in B}(x_{ij} - l_{ij})\right\}$$

# Klein's algorithm.

1. Let $x$ be a feasible flow.
2. **while** exist augmenting cycle $C$ **do**
3. $x := x + \gamma_C^{\delta(C)}$
4. **end while**
5. **return** $x$

# Residual Network

- Analogous to residual network used in the Ford-Fulkerson max-flow algorithm – represents possible changes to flow on arcs maintaining feasibility.

- Definition is straightforward given feasible flow $x$:
  - Same set of nodes.
  - Circulation network.
  - Upper bounds, *residual capacities*.

- When defined correctly, *directed cycles* in residual network $D_x$ correspond exactly to augmenting cycles for the flow $x$.

# Partial correctness

- **Circulation decomposition lemma**: Let $x \geq 0$ be circulation in network $D$. Then there are cycles $C_1, \dots, C_k$ and $\delta_1, \dots, \delta_k > 0$ such that

$$x = \gamma_{C_1}^{\delta_1} + \cdots \gamma_{C_k}^{\delta_k}$$

Apply to residual network $D_x$:

The difference $x - y$ between the flow $x$ and any optimal flow $y$ can be expressed as a feasible flow in $D_x$. If $x$ is not optimal then there must be a cycle in $D_x$.

# Initialization, Termination, Complexity,…

- Termination:
  - Cost is strictly improved each iteration. When balances, lower and upper bounds are integers, this guarantees termination.

- Complexity:
  - May take exponentially many iterations (like Ford-Fulkerson).

- Initialization:
  - Formulate finding feasible flow as a maximum flow problem:
  - Arcs for nodes $i \in N$:
    - From $s$ to $i$ with capacity $b_i$ when $b_i > 0$.
    - From $i$ to $t$ with capacity $-b_i$ when $b_i < 0$.
  - Arcs for arcs $ij \in A$: