Assignment Problem

Assignment problem.
- Input: weighted, complete bipartite graph $G = (L \cup R, E)$ with $|L| = |R|$.
- Goal: find a perfect matching of min weight.

<table>
<thead>
<tr>
<th></th>
<th>1'</th>
<th>2'</th>
<th>3'</th>
<th>4'</th>
<th>5'</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>8</td>
<td>9</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>10</td>
<td>7</td>
<td>16</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>13</td>
<td>11</td>
<td>19</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>13</td>
<td>12</td>
<td>20</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>7</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>

Min cost perfect matching
$M = \{1-2', 2-3', 3-5', 4-1', 5-4'\}$
$\text{cost}(M) = 8 + 7 + 10 + 8 + 11 = 44$

Applications

Natural applications.
- Match jobs to machines.
- Match personnel to tasks.
- Match PU students to writing seminars.

Non-obvious applications.
- Vehicle routing.
- Signal processing.
- Virtual output queueing.
- Multiple object tracking.
- Approximate string matching.
- Enhance accuracy of solving linear systems of equations.
Bipartite Matching

Bipartite matching. Can solve via reduction to max flow.

Flow. During Ford-Fulkerson, all capacities and flows are 0/1. Flow corresponds to edges in a matching $M$.

Residual graph $G_M$ simplifies to:
- If $(x, y) \notin M$, then $(x, y)$ is in $G_M$.
- If $(x, y) \in M$, the $(y, x)$ is in $G_M$.

Augmenting path simplifies to:
- Edge from $s$ to an unmatched node $x \in X$.
- Alternating sequence of unmatched and matched edges.
- Edge from unmatched node $y \in Y$ to $t$.

Assignment Problem: Successive Shortest Path Algorithm

Cost of an alternating path. Pay $c(x, y)$ to match $x-y$; receive $c(x, y)$ to unmatch $x-y$.

Shortest alternating path. Alternating path from any unmatched node $x \in X$ to any unmatched node $y \in Y$ with smallest cost.

Successive shortest path algorithm.
- Start with empty matching.
- Repeatedly augment along a shortest alternating path.

Finding The Shortest Alternating Path

Shortest alternating path. Corresponds to shortest $s$-$t$ path in $G_M$.

Concern. Edge costs can be negative.

Fact. If always choose shortest alternating path, then $G_M$ contains no negative cycles $\Rightarrow$ compute using Bellman-Ford.

Our plan. Use duality to avoid negative edge costs (and negative cost cycles) $\Rightarrow$ compute using Dijkstra.
**Duality intuition.** Adding (or subtracting) a constant to every entry in row \( x \) or column \( y \) does not change the min cost perfect matching(s).

**Reduced Costs**

**Reduced costs.** For \( x \in X, y \in Y \), define \( c^p(x, y) = p(x) + c(x, y) - p(y) \).

**Observation 1.** Finding a min cost perfect matching with reduced costs is equivalent to finding a min cost perfect matching with original costs.

**Compatible Prices**

**Compatible prices.** For each node \( v \), maintain prices \( p(v) \) such that:
- (i) \( c^p(x, y) \geq 0 \) for all \( (x, y) \not\in M \).
- (ii) \( c^p(x, y) = 0 \) for all \( (x, y) \in M \).

**Observation 2.** If \( p \) are compatible prices for a perfect matching \( M \), then \( M \) is a min cost perfect matching.
Successive shortest path.

**Successive Shortest Path Algorithm**

\[ \text{Successive-Shortest-Path}(X, Y, c) \]

\[
M \leftarrow \emptyset \\
\text{foreach } x \in X: \quad p(x) \leftarrow 0 \\
\text{foreach } y \in Y: \quad p(y) \leftarrow \min_e \text{ into } y \quad c(e) \\
\text{while } (M \text{ is not a perfect matching}) \quad \{
\quad \text{Compute shortest path distances } d \\
\quad P \leftarrow \text{shortest alternating path using costs } c^p \\
\quad M \leftarrow \text{updated matching after augmenting along } P \\
\quad \text{foreach } v \in X \cup Y: \quad p(v) \leftarrow p(v) + d(v) \\
\} \\
\text{return } M
\]

**Lemma 1.** Let \( p \) be compatible prices for matching \( M \). Let \( d \) be shortest path distances in \( G_M \) with costs \( c^p \). All edges \((x, y)\) on shortest path have \( c^{p+d}(x, y) = 0 \).

**Proof.**
- If \((x, y) \in M\), then \((y, x)\) is the only edge entering \( x \) in \( G_M \). Thus, \((y, x)\) on shortest path.
- By Lemma 1, \( c^{p+d}(x, y) = 0 \).

**Proof.**
- If \((x, y) \notin M\), then \((x, y)\) is an edge in \( G_M \).
- Since we augment along a min cost path, the only edges \((x, y)\) that swap into or out of the matching are on the shortest path.
- By Lemma 1, these edges satisfy \( c^{p+d}(x, y) = 0 \).
- Thus, compatibility is maintained.

**Lemma 2.** Let \( p \) be compatible prices for matching \( M \). Let \( d \) be shortest path distances in \( G_M \) with costs \( c^p \). Then \( p' = p + d \) are also compatible prices for \( M \).

**Proof.**
- By Lemma 1, the prices \( p + d \) are compatible for \( M \).
- Since we augment along a min cost path, the only edges \((x, y)\) that swap into or out of the matching are on the shortest path.
- By Lemma 1, these edges satisfy \( c^{p+d}(x, y) = 0 \).
- Thus, compatibility is maintained.

**Lemma 3.** Let \( M' \) be matching obtained by augmenting along a min cost path with respect to \( c^{p+d} \). Then \( p' = p + d \) is compatible with \( M' \).

**Proof.**
- By Lemma 2, the prices \( p + d \) are compatible for \( M' \).
- Since we augment along a min cost path, the only edges \((x, y)\) that swap into or out of the matching are on the shortest path.
- By Lemma 1, these edges satisfy \( c^{p+d}(x, y) = 0 \).
- Thus, compatibility is maintained.

**Reduced costs:**
\[ c^p(x, y) = p(x) + c(x, y) - p(y). \]

\textbf{Pf.} Follows from Lemmas 2 and 3 and initial choice of prices.

\textbf{Theorem.} The algorithm returns a min cost perfect matching.

\textbf{Pf.} Upon termination $M$ is a perfect matching, and $p$ are compatible prices. Optimality follows from Observation 2.

\textbf{Theorem.} The algorithm can be implemented in $O(n^3)$ time.

\textbf{Pf.}
\begin{itemize}
  \item Each iteration increases the cardinality of $M$ by 1 $\Rightarrow$ $n$ iterations.
  \item Bottleneck operation is computing shortest path distances $d$.
  \item Since all costs are nonnegative, each iteration takes $O(n^2)$ time using (dense) Dijkstra.
\end{itemize}

\textbf{Weighted Bipartite Matching}

\textbf{Weighted bipartite matching.} Given weighted bipartite graph, find maximum cardinality matching of minimum weight.

\textbf{Successive shortest path algorithm.} $O(mn \log n)$ time using heap-based version of Dijkstra’s algorithm.

\textbf{Best known bounds.} $O(mn^{1/2})$ deterministic; $O(n^{2.376})$ randomized.

\textbf{Planar weighted bipartite matching.} $O(n^{3/2} \log^5 n)$.

\textbf{Input-Queued Switching}

\textbf{Input-queued switch.}
\begin{itemize}
  \item $n$ inputs and $n$ outputs in an $n$-by-$n$ crossbar layout.
  \item At most one cell can depart an input at a time.
  \item At most one cell can arrive at an output at a time.
  \item Cell arrives at input $x$ and must be routed to output $y$.
\end{itemize}
Input-Queued Switching

**FIFO queueing.** Each input $x$ maintains one queue of cells to be routed.

**Head-of-line blocking (HOL).**
- A cell can be blocked by a cell queued ahead of it that is destined for a different output.
- Can limit throughput to 58%, even when arrivals are uniform.

Maximum size matching. Find a max cardinality matching.
- Achieves 100% when arrivals are uniform.
- Can starve input-queues when arrivals are non-uniform.

Virtual output queueing (VOQ). Each input $x$ maintains $n$ queue of cells, one for each output $y$.

**Max weight matching.** Find a min cost perfect matching between inputs $x$ and outputs $y$, where $c(x, y)$ equals:
- [LQF] The number of cells waiting to go from input $x$ to output $y$.
- [OCF] The waiting time of the cell at the head of VOQ from $x$ to $y$.

**Theorem.** LQF and OCF achieve 100% throughput if arrivals are independent.

**Practice.**
- Too slow in practice for this application; difficult to implement in hardware. Provides theoretical framework.
- Use maximal (weighted) matching $\Rightarrow$ 2-approximation.

Reference: [http://robotics.eecs.berkeley.edu/~wlr/Papers/AMMW.pdf](http://robotics.eecs.berkeley.edu/~wlr/Papers/AMMW.pdf)