7. **Network Flow III**

- assignment problem
- input-queued switching
7. **Network Flow III**

- assignment problem
- input-queued switching

**Section 7.13**
Assignment problem

Input. Weighted, complete bipartite graph $G = (X \cup Y, E)$ with $|X| = |Y|$.  
Goal. Find a perfect matching of min weight.
Assignment problem

**Input.** Weighted, complete bipartite graph $G = (X \cup Y, E)$ with $|X| = |Y|$.

**Goal.** Find a perfect matching of min weight.

---

**min-cost perfect matching**

$M = \{ 0-2', 1-0', 2-1' \}$

$\text{cost}(M) = 3 + 5 + 4 = 12$
Princeton writing seminars

Goal. Given $m$ seminars and $n = 12m$ students who rank their top 8 choices, assign each student to one seminar so that:
  - Each seminar is assigned exactly 12 students.
  - Students tend to be "happy" with their assigned seminar.

Solution.
  - Create one node for each student $i$ and 12 nodes for each seminar $j$.
  - Solve assignment problem where $c_{ij}$ is some function of the ranks:

$$c_{ij} = \begin{cases} f(rank(i, j)) & \text{if } i \text{ ranks } j \\ \infty & \text{if } i \text{ does not rank } j \end{cases}$$
Locating objects in space

**Goal.** Given $n$ objects in 3d space, locate them with 2 sensors.

**Solution.**

- Each sensor computes line from it to each particle.
- Let $c_{ij} =$ distance between line $i$ from sensor 1 and line $j$ from sensor 2.
- Due to measurement errors, we might have $c_{ij} > 0$.
- Solve assignment problem to locate $n$ objects.
Kidney exchange

If a donor and recipient have a different blood type, they can exchange their kidneys with another donor and recipient pair in a similar situation.

Can also be done among multiple pairs (or starting with an altruistic donor).
Kidney exchange

weight = 3 + 5 + 7 + 8 + 4 = 27

weight = 2 + 5 + 8 + 6 + 1 + 9 = 31
Applications

Natural applications.

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

Non-obvious applications.

- Vehicle routing.
- Kidney exchange.
- Signal processing.
- Multiple object tracking.
- Virtual output queueing.
- Handwriting recognition.
- Locating objects in space.
- Approximate string matching.
- Enhance accuracy of solving linear systems of equations.
Bipartite matching

Bipartite matching. Can solve via reduction to maximum flow.

Flow. During Ford-Fulkerson, all residual 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 $(x, y) \in M$, then $(y, x)$ 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 $t$. 
**Alternating path**

**Def.** An alternating path $P$ with respect to a matching $M$ is an alternating sequence of unmatched and matched edges, starting from an unmatched node $x \in X$ and going to an unmatched node $y \in Y$.

**Key property.** Can use $P$ to increase by one the cardinality of the matching.

**Pf.** Set $M' = M \oplus P$.

![Diagram](image.png)
Assignment problem: successive shortest path algorithm

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

![Diagram of alternating paths](image)

- \( P = 2 \to 2' \to 1 \to 1' \)
- \( \text{cost}(P) = 2 - 6 + 10 = 6 \)

**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 minimum cost $s \rightarrow t$ path in $G_M$.

![Graph](image)

**Concern.** Edge costs can be negative.

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

**Our plan.** Use duality to avoid negative edge costs (and negative cycles) $\Rightarrow$ can compute using Dijkstra.
Equivalent assignment problem

**Duality intuition.** Adding a constant $p(x)$ to the cost of every edge incident to node $x \in X$ does not change the min-cost perfect matching(s).

**Pf.** Every perfect matching uses exactly one edge incident to node $x$. ■

---

**original costs** $c(x, y)$

- $c(0, 0') = 18$
- $c(0, 1) = 10$
- $c(0, 1') = 6$
- $c(1, 0) = 6$
- $c(1, 1') = 2$
- $c(1, 2) = 9$
- $c(1, 2') = 4$
- $c(2, 0) = 5$
- $c(2, 1) = 6$
- $c(2, 1') = 2$
- $c(2, 2') = 9$
- $c(0, 0') = 15$
- $c(0, 1) = 7$
- $c(0, 1') = 3$
- $c(1, 0) = 5$
- $c(1, 1') = 6$
- $c(1, 2) = 2$
- $c(1, 2') = 4$
- $c(2, 0) = 3$
- $c(2, 1) = 6$
- $c(2, 1') = 2$
- $c(2, 2') = 4$

**modified costs** $c'(x, y)$

- $c'(0, 0') = 18$
- $c'(0, 1) = 10$
- $c'(0, 1') = 6$
- $c'(1, 0) = 6$
- $c'(1, 1') = 2$
- $c'(1, 2) = 9$
- $c'(1, 2') = 4$
- $c'(2, 0) = 5$
- $c'(2, 1) = 6$
- $c'(2, 1') = 2$
- $c'(2, 2') = 9$
- $c'(0, 0') = 15$
- $c'(0, 1) = 7$
- $c'(0, 1') = 3$
- $c'(1, 0) = 5$
- $c'(1, 1') = 6$
- $c'(1, 2) = 2$
- $c'(1, 2') = 4$
- $c'(2, 0) = 3$
- $c'(2, 1) = 6$
- $c'(2, 1') = 2$
- $c'(2, 2') = 4$

$p(0) = 3$

---

**Note:**

- Add 3 to all edges incident to node 0.
Equivalent assignment problem

Duality intuition. Subtracting a constant \( p(y) \) to the cost of every edge incident to node \( y \in Y \) does not change the min-cost perfect matching(s).

Pf. Every perfect matching uses exactly one edge incident to node \( y \).  

\[ \text{original costs } c(x, y) \]

\[ \text{modified costs } c'(x, y) \]

\[ p(0') = 5 \]

substitute 5 from all edges incident to node 0'
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 $v \in X \cup Y$, maintain prices $p(v)$ such that:

- $c_p(x, y) \geq 0$ for all $(x, y) \notin M$.
- $c_p(x, y) = 0$ for all $(x, y) \in M$.

Observation 2. If prices $p$ are compatible with a perfect matching $M$, then $M$ is a min-cost perfect matching.

Pf. Matching $M$ has 0 cost. □
Successive shortest path algorithm

**SUCCESSIVE-SHORTEST-PATH** \((X, Y, c)\)

\[ M \leftarrow \emptyset. \]

**FOREACH** \(v \in X \cup Y : p(v) \leftarrow 0.\) \(\text{prices } p \text{ are compatible with } M\)

\[ c^p(x, y) = c(x, y) \geq 0 \]

**WHILE** (\(M\) is not a perfect matching)

\[ d \leftarrow \text{shortest path distances using costs } c^p. \]

\[ P \leftarrow \text{shortest alternating path using costs } c^p. \]

\[ M \leftarrow \text{updated matching after augmenting along } P. \]

**FOREACH** \(v \in X \cup Y : p(v) \leftarrow p(v) + d(v). \)

**RETURN** \(M.\)
Successive shortest path algorithm

Initialization.

- $M = \emptyset$.
- For each $v \in X \cup Y : p(v) \leftarrow 0$. 
Successive shortest path algorithm

Initialization.

• \( M = \emptyset \).
• For each \( v \in X \cup Y : p(v) \leftarrow 0 \).
Successive shortest path algorithm

Step 1.
- Compute shortest path distances $d(v)$ from $s$ to $v$ using $c^p(x, y)$.
- Update matching $M$ via shortest path from $s$ to $t$.
- For each $v \in X \cup Y$: $p(v) \leftarrow p(v) + d(v)$. 

![Diagram showing shortest path algorithm]

- $d(0) = 0$
- $d(1) = 0$
- $d(2) = 0$
- $d(0') = 5$
- $d(1') = 4$
- $d(2') = 1$
- $d(s) = 0$
- $d(1) = 0$
- $d(2) = 0$
- $d(t) = 1$
- $d(0') = 5$
- $d(1') = 4$
- $d(2') = 1$
Successive shortest path algorithm

Step 1.
- Compute shortest path distances $d(v)$ from $s$ to $v$ using $c^p(x, y)$.
- Update matching $M$ via shortest path from $s$ to $t$.
- For each $v \in X \cup Y$: $p(v) \leftarrow p(v) + d(v)$.
**Successive shortest path algorithm**

**Step 1.**
- Compute shortest path distances $d(v)$ from $s$ to $v$ using $c^p(x, y)$.
- Update matching $M$ via shortest path from $s$ to $t$.
- For each $v \in X \cup Y$: $p(v) \leftarrow p(v) + d(v)$.

**Diagram:**

- **Vertices:** $s$, $0$, $1$, $2$, $0'$, $1'$, $2'$, $t$
- **Edges and Costs:**
  - From $s$ to $1$: $0$
  - From $s$ to $2$: $0$
  - From $0$ to $0'$: $10$
  - From $0$ to $1$: $3$
  - From $0$ to $2$: $2$
  - From $1$ to $0'$: $0$
  - From $1$ to $1'$: $2$
  - From $1$ to $2$: $1$
  - From $1'$ to $0$: $2$
  - From $1'$ to $1$: $1$
  - From $1'$ to $2'$: $4$
  - From $2$ to $0'$: $4$
  - From $2$ to $1'$: $0$
  - From $2$ to $2'$: $0$

**Initial Matching:**
- $p(0) = 0$
- $p(1) = 0$
- $p(2) = 0$
- $p(0') = 5$
- $p(1') = 4$
- $p(2') = 1$

**Reduced Costs $c^p(x, y)$**

**Matching:**
- $2 \rightarrow 2'$
**Successive shortest path algorithm**

Step 2.
- Compute shortest path distances $d(v)$ from $s$ to $v$ using $c^p(x, y)$.
- Update matching $M$ via shortest path from $s$ to $t$.
- For each $v \in X \cup Y$: $p(v) \leftarrow p(v) + d(v)$.

![Diagram of shortest path algorithm]

- $d(0) = 0$
- $d(1) = 0$
- $d(2) = 1$
- $d(0') = 0$
- $d(1') = 1$
- $d(2') = 1$
- $d(t) = 0$

**Matching**

$2 \rightarrow 2'$

**Shortest path distances**

$\begin{align*}
    d(0) &= 0 \\
    d(1) &= 0 \\
    d(2) &= 1 \\
    d(0') &= 0 \\
    d(1') &= 1 \\
    d(2') &= 1 \\
    d(t) &= 0
\end{align*}$
Successive shortest path algorithm

Step 2.
- Compute shortest path distances $d(v)$ from $s$ to $v$ using $c^p(x, y)$.
- Update matching $M$ via shortest path from $s$ to $t$.
- For each $v \in X \cup Y$: $p(v) \leftarrow p(v) + d(v)$. 

![Diagram](image-url)
Successive shortest path algorithm

Step 2.

- Compute shortest path distances $d(v)$ from $s$ to $v$ using $c^p(x, y)$.
- Update matching $M$ via shortest path from $s$ to $t$.
- For each $v \in X \cup Y$: $p(v) \leftarrow p(v) + d(v)$.

![Graph diagram showing the successive shortest path algorithm with node labels and edge weights.](image)
Successive shortest path algorithm

Step 3.
- Compute shortest path distances $d(v)$ from $s$ to $v$ using $c^p(x, y)$.
- Update matching $M$ via shortest path from $s$ to $t$.
- For each $v \in X \cup Y$: $p(v) \leftarrow p(v) + d(v)$.

![Diagram showing shortest path distances and matching]
Step 3.

- Compute shortest path distances $d(v)$ from $s$ to $v$ using $c_p(x, y)$.
- Update matching $M$ via shortest path from $s$ to $t$.
- For each $v \in X \cup Y$: $p(v) \leftarrow p(v) + d(v)$.
Successive shortest path algorithm

Step 3.

- Compute shortest path distances $d(v)$ from $s$ to $v$ using $c^p(x, y)$.
- Update matching $M$ via shortest path from $s$ to $t$.
- For each $v \in X \cup Y$: $p(v) \leftarrow p(v) + d(v)$.
Successive shortest path algorithm

Termination.

- $M$ is a perfect matching.
- Prices $p$ are compatible with $M$. 

![Graph showing the Successive shortest path algorithm with reduced costs $c^p(x, y)$ and matching 1-0' 0-2' 2-1'.]
**Maintaining compatible prices**

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

**Pf.** Let $(x, y)$ be some edge on shortest path.

- If $(x, y) \in M$, then $(y, x)$ on shortest path and $d(x) = d(y) - c_p(x, y)$;
- If $(x, y) \notin M$, then $(x, y)$ on shortest path and $d(y) = d(x) + c_p(x, y)$.
- In either case, $d(x) + c_p(x, y) - d(y) = 0$.
- By definition, $c_p(x, y) = p(x) + c(x, y) - p(y)$.
- Substituting for $c_p(x, y)$ yields $(p(x) + d(x)) + c(x, y) - (p(y) + d(y)) = 0$.
- In other words, $c_{p+d}(x, y) = 0$.  □

Given prices $p$, the reduced cost of edge $(x, y)$ is

$$c_p(x, y) = p(x) + c(x, y) - p(y).$$
Maintaining compatible prices

Lemma 2. Let $p$ be compatible prices for $M$. Let $d$ be shortest path distances in $G_M$ with costs $c_p$. Then $p' = p + d$ are also compatible prices for $M$.

Pf. $(x, y) \in M$

- $(y, x)$ is the only edge entering $x$ in $G_M$. Thus, $(y, x)$ on shortest path.
- By Lemma 1, $c_p + d(x, y) = 0$.

Pf. $(x, y) \not\in M$

- $(x, y)$ is an edge in $G_M \Rightarrow d(y) \leq d(x) + c_p(x, y)$.
- Substituting $c_p(x, y) = p(x) + c(x, y) - p(y) \geq 0$ yields 
  
  $$(p(x) + d(x)) + c(x, y) - (p(y) + d(y)) \geq 0.$$  
- In other words, $c_p + d(x, y) \geq 0$. 

Prices $p$ are compatible with matching $M$:

- $c_p(x, y) \geq 0$ for all $(x, y) \not\in M$.
- $c_p(x, y) = 0$ for all $(x, y) \in M$. 

Maintaining compatible prices

**Lemma 3.** Let $p$ be compatible prices for $M$ and let $M'$ be matching obtained by augmenting along a min cost path with respect to $c^p + d$. Then $p' = p + d$ are compatible prices for $M'$.

**Pf.**

- 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 min-cost path.
- By **Lemma 1**, these edges satisfy $c^p + d(x, y) = 0$.
- Thus, compatibility is maintained. •

Prices $p$ are compatible with matching $M$:

- $c^p(x, y) \geq 0$ for all $(x, y) \notin M$.
- $c^p(x, y) = 0$ for all $(x, y) \in M$. 
Successive shortest path algorithm: analysis

Invariant. The algorithm maintains a matching \( M \) and compatible prices \( p \).
Pf. Follows from Lemma 2 and Lemma 3 and initial choice of prices. ▪

Theorem. The algorithm returns a min-cost perfect matching.
Pf. Upon termination \( M \) is a perfect matching, and \( p \) are compatible prices. Optimality follows from Observation 2. ▪

Theorem. The algorithm can be implemented in \( O(n^3) \) time.
Pf.
• Each iteration increases the cardinality of \( M \) by 1 \( \Rightarrow \) \( n \) iterations.
• Bottleneck operation is computing shortest path distances \( d \).
  Since all costs are nonnegative, each iteration takes \( O(n^2) \) time
  using (dense) Dijkstra. ▪
Weighted bipartite matching

Weighted bipartite matching. Given a weighted bipartite graph with \( n \) nodes and \( m \) edges, find a maximum cardinality matching of minimum weight.

Theorem. [Fredman-Tarjan 1987] The successive shortest path algorithm solves the problem in \( O(n^2 + mn \log n) \) time using Fibonacci heaps.

Theorem. [Gabow-Tarjan 1989] There exists an \( O(mn^{1/2} \log(nC)) \) time algorithm for the problem when the costs are integers between 0 and \( C \).
History

Thorndike 1950. Formulated in a modern way by a psychologist.

THE PROBLEM OF CLASSIFICATION OF PERSONNEL

ROBERT L. THORNDIKE
TEACHERS COLLEGE, COLUMBIA UNIVERSITY

The personnel classification problem arises in its pure form when all job applicants must be used, being divided among a number of job categories. The use of tests for classification involves problems of two types: (1) problems concerning the design, choice, and weighting of tests into a battery, and (2) problems of establishing the optimum administrative procedure of using test results for assignment. A consideration of the first problem emphasizes the desirability of using simple, factorially pure tests which may be expected to have a wide range of validities for different job categories. In the use of test results for assignment, an initial problem is that of expressing predictions of success in different jobs in comparable score units. These units should take account of predictor validity and of job importance. Procedures are described for handling assignment either in terms of daily quotas or in terms of a stable predicted yield.

Assign individuals to jobs to maximize average success of all individuals.
History

Thorndike 1950. Formulated in a modern way by a psychologist.

There are, as has been indicated, a finite number of permutations in the assignment of men to jobs. When the classification problem as formulated above was presented to a mathematician, he pointed to this fact and said that from the point of view of the mathematician there was no problem. Since the number of permutations was finite, one had only to try them all and choose the best. He dismissed the problem at that point. This is rather cold comfort to the psychologist, however, when one considers that only ten men and ten jobs mean over three and a half million permutations. Trying out all the permutations may be a mathematical solution to the problem, it is not a practical solution.

anticipated theory of computational complexity!
History

Kuhn 1955. First poly-time algorithm; named "Hungarian" algorithm to honor two Hungarian mathematicians (Kőnig and Egerváry).

Munkres 1957. Reviewed algorithm; observed $O(n^4)$ implementation.

Edmonds-Karp, Tomizawa 1971. Improved to $O(n^3)$.

---

THE HUNGARIAN METHOD FOR THE ASSIGNMENT PROBLEM

H. W. Kuhn
Bryn Mawr College

Assuming that numerical scores are available for the performance of each of n persons on each of n jobs, the "assignment problem" is the quest for an assignment of persons to jobs so that the sum of the n scores so obtained is as large as possible. It is shown that ideas latent in the work of two Hungarian mathematicians may be exploited to yield a new method of solving this problem.

anticipated development of combinatorial optimization
History

Jacobi (1804-1851). Introduces a bound on the order of a system of \( m \) ordinary differential equations in \( m \) unknowns and reduces it to....

Looking for the order of a system of arbitrary ordinary differential equations
History

Jacobi (1804-1851). The assignment problem! Moreover, he provides a polynomial-time algorithm.

Jacobi formulated the assignment problem; proposed and analyzed the Hungarian algorithm.
7. **Network Flow III**

- assignment problem
- input-queued switching
Input-queued switching

Input-queued switch.
- $n$ input ports and $n$ output ports in an $n$-by-$n$ crossbar layout.
- At most one cell can depart an input at a time.
- At most one cell can arrive at an output at a time.
- Cell arrives at input $x$ and must be routed to output $y$.

Application. High-bandwidth switches.
FIFO queuing

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.
FIFO queuing

FIFO queuing. 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.

Fact. FIFO can limit throughput to 58% even when arrivals are uniform i.i.d.
Virtual output queueing

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

Maximum size matching. Find a max cardinality matching.

Fact. VOQ achieves 100% throughput when arrivals are uniform i.i.d. but can starve input-queues when arrivals are nonuniform.
Input-queued switching

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 (even if not uniform).

Practice.

- Assignment problem too slow in practice.
- Difficult to implement in hardware.
- Provides theoretical framework: use maximal (weighted) matching.

Achieving 100% Throughput in an Input-Queued Switch

Nick McKeown, Senior Member, IEEE, Adisak Mekkittikul, Member, IEEE, Venkat Anantharam, Fellow, IEEE, and Jean Walrand, Fellow, IEEE

Abstract—It is well known that head-of-line blocking limits the throughput of an input-queued switch with first-in-first-out (FIFO) queues. Under certain conditions, the throughput can be shown to be limited to approximately 58.6%. It is also known that if non-FIFO queuing policies are used, the throughput can be increased. However, it has not been previously shown that if a suitable queuing policy and scheduling algorithm are used, then it is possible to achieve 100% throughput for all independent arrival processes. In this paper we prove this to be the case using a simple linear programming argument and quadratic Lyapunov function. In particular, we assume that each input maintains a separate FIFO queue for each output and that the switch is scheduled using a maximum weight bipartite matching algorithm. We introduce two maximum weight matching algorithms: longest queue first (LQF) and oldest cell first (OCF). Both algorithms achieve 100% throughput for all independent arrival processes. LQF favors queues with larger occupancy, ensuring that larger queues will eventually be served. However, we find that LQF can lead to the permanent starvation of short queues. OCF overcomes this limitation by favoring cells with large waiting times.

Index Terms—Arbitration, ATM, input-queued switch, input-queuing, packet switch, queuing networks, scheduling algorithm.