Outline

Matching

Flow networks

Third-year projects

The stable marriage problem
• Consider the following (slightly idealized) matchmaking problem.
• We are given:
  • a set of \( n \) boys and \( n \) girls;
  • a specification of who is prepared to marry whom.
• We want to compute:
  • a 1–1 pairing of boys with girls producing the maximum number of couples.
• For example:
• For example:
• For example:
• For example:
For example:
• For example:
• For example:
For example:
A bipartite graph is a triple $G = (U, V, E)$ where $U$ and $V$ are disjoint sets and $E \subseteq U \times V$.

Note that $G = (U \cup V, E)$ can be regarded as a (directed) graph in the sense of the previous lecture.
Let \( G = (V, W, E) \) be a bipartite graph. A matching is a subset \( E' \subseteq E \) such that for all \( v \in V \), there is at most one \( w \in W \) with \( (v, w) \in E' \), and, for all \( w \in W \), there is at most one \( v \in V \) with \( (v, w) \in E' \).

The matching is perfect if every node in \( V \) and \( W \) is incident to some \( e \in E' \).

We then have the problem:

**MATCHING**

Given: A bipartite Graph \( G \)

Return: Yes if \( G \) has a perfect matching, and No otherwise.
Here is a naïve algorithm for solving this problem

begin naiveMatch(V, W, E)
    if V = ∅_
        if W = ∅ return Yes
        else return No
    pick v ∈ V
    for each {w | (v, w) ∈ E} _
        Let (V′, W′, E′) be result of removing v, w from (V, W, E)_
        if naiveMatch(V′, W′, E′), return Yes
    return No

It is pretty clear that this will in general take exponential time in \( n = |V| \).

Maybe we can do better . . .
Outline

Matching

Flow networks

Third-year projects

The stable marriage problem
• Let \((V, E)\) be a directed graph. A back-loop is a pair of edges \(((u, v), (v, u))\).

• A flow network is a quintuple \((V, E, s, t, c)\) where \((V, E)\) is a directed graph with no back-loops, \(s, t \in V\) are distinct, and \(c : E \to \mathbb{N}\).

• We assume \(s\) has no incoming edges and \(t\) no outgoing edges.
Let \( N = (V, E, s, t, c) \) be a flow network. A flow in \( N \) is a function \( f : E \rightarrow \mathbb{R}^+ \) with the following properties:

- The net flow out of any node \( u \in (V \setminus \{s, t\}) \) is zero:
  \[
  \sum_{v : (u, v) \in E} f(u, v) - \sum_{v : (v, u) \in E} f(v, u) = 0.
  \]

- No edge \((u, v) \in E\) exceeds its capacity:
  \[
  f(u, v) \leq c(u, v).
  \]

- The value of \( f \) is the quantity:
  \[
  \sum_{v : (s, v) \in E} f(s, v) - \sum_{v : (v, s) \in E} f(v, s) = \sum_{(v, t) \in E} f(v, t) - \sum_{v : (t, v) \in E} f(t, v).
  \]
• Our problem: given a flow network $N$, compute a flow $f$ for $N$ whose value is maximum; such a flow will be called optimal.

• That is, we have the computation task:

\[
\text{MAX FLOW} \\
\text{Given: A flow network } N \text{ } \\
\text{Return: An optimal flow } f \text{ for } N.
\]

• It is obvious that such an optimal flow exists. Notice that it will not in general be unique.
Here is a flow (numbers in red) for the above flow network:
• The key idea: given a flow network \( N = (V, E, s, t, c) \) and a flow \( f \), we construct an auxiliary directed graph \( N_f \), \( G = (V, E_f) \), where \( E_f \) is the set of pairs:

\[
\{(u, v) \in E \mid f(u, v) < c(u, v)\} \cup \\
\{(u, v) \mid (v, u) \in E \text{ and } f(v, u) > 0\}.
\]

• That is, the edges of directed graph \( N_f \) are
  • those edges in \( E \) on which the capacity is not exhausted;
  • the reversal of those edges in \( E \) on which there is some flow.

• Note that \( N_f \) may contain back-loops.
- **Original network** $N$ with flow $f$:

```
Original network N with flow f:
```

```

```

- **Auxiliary directed graph** $N_f$:

```
Auxiliary directed graph N_f:
```

```
```
• Original network $N$ with flow $f$:

• Auxiliary directed graph $N_f$:
• **Original network** $N$ with new flow $f'$:

```
1 (1)
5 (5) 13 (4) 8 (8) 9 (9)
14 (10) 2 (2)
11 (4)
```

• **New auxiliary directed graph** $N_{f'}$:
Lemma (Min-cut, Max-flow)

Let \( N = (V, E, s, t, c) \) be a flow network and \( f \) a flow in \( N \). Then there is a path in \( N_f \) from \( s \) to \( t \) if and only if \( f \) is not optimal.

Proof.

The only-if direction is trivial. Suppose, conversely, there is no path from \( s \) to \( t \) in \( N_f \). Let \( S \) be the set of nodes \( V \) reachable from \( s \) in \( N_f \), and \( T = V \setminus S \).

There is no backflow from \( T \) to \( S \), and no spare capacity from \( S \) to \( T \).
Here is the algorithm:

begin flowMax\((V,E,s,t,c)\)
set \(f(e) = 0\) for all \(e \in E\)
while reachable\((N_f,s,t)\)
let \(e_1, \ldots, e_m\) be the edges on a path from \(s\) to \(t\)
for \(1 \leq i \leq m\)
if \(e_i \in E\), increment \(f(e_i)\)
else decrement \(f(e_i^{-1})\)
return \(f\)

By the Min-Cut-Max-Flow Theorem, this algorithm terminates with \(f\) optimal.

This argument has the following amazing corollary:

*If a flow network has integral capacities, then there is an optimal flow with integral values.*
• suppose $|V| = n$ and $c(e) \leq C$.
• We know that REACHABILITY in $N_f$ can be solved in $\text{TIME}(O(n + m))$ where $n$ is the number of nodes and $m$ the number of edges.
• The maximum flow is $nC$ ($s$ is connected to $< n$ other nodes, with capacity $\leq C$, so there are at most $nC$ cycles of the main loop).
• Hence the operating time is at $O(n(n + m)C)$.
• (Actually, we can do a bit better in regard to $C$ . . . )
A flow network with costs is a sextuple $N = (V, E, s, t, c, \gamma)$ where $(V, E, s, t, c)$ is a flow network and $\gamma : E \rightarrow \mathbb{N}$.

We think of $\gamma$ as a cost function, where $\gamma(e)$ is the price per unit of flow along $e$.

We can draw a flow network with each edge $e$ labelled $c(e)/\gamma(e)$.

The total cost of a flow $f$ in $N$ is $\sum_{e \in E} f(e) \cdot \gamma(e)$. 
• This suggests another interesting computation task:

**MAX FLOW**

Given: A flow network with cost function, $N$
Return: An optimal flow $f$ for $N$ having the minimum cost among all optimal flows.

• It is again obvious that such a minimum cost optimal flow exists. Again, it will not in general be unique.
• In fact, a slight modification of the above algorithm will solve this problem beautifully.
• We regard the cost function as giving distances in the derived graph:
  • if \((u, v)\) is an edge of \(N_f\) and \((u, v) \in E\), the distance \(d(u, v)\) is \(\gamma(e)\);
  • if \((u, v)\) is an edge of \(N_f\) and \((v, u) \in E\), the distance \(d(u, v)\) is \(-\gamma(e)\).
• Notice that there these ‘distances’ may be negative.
• However, we may assume that there are no cycles of total negative length. (This is actually rather tricky to see . . . )
Lemma

Let $N$ be a flow network with cost function, and suppose $f$ is a flow of value $v$ through $N$, such that $f$ has minimal cost among all flows of value $v$. Suppose $\pi$ is a path in $N_f$ from $s$ to $t$ of minimal length, and let $f'$ be obtained from $f$ by augmenting along $\pi$ (in the usual way). Then $f'$ has minimal cost among all flows of value $v + 1$.

- A rigorous proof is quite involved, and we will not give one in this course.
Here is the resulting algorithm for computing minimum cost optimal flows:

begin flowMaxCost(\(V,E,s,t,c,\gamma\))

set \(f(e) = 0\) for all \(e \in E\)

while reachable(\(N_f,s,t\))

let \(e_1, \ldots, e_m\) be the edges on a path from \(s\) to \(t\) of minimal length according to \(\gamma\)

for \(1 \leq i \leq m\)

if \(e_i \in E\), increment \(f(e_i)\)
else decrement \(f(e_i^{-1})\)

return \(f\)

It is sometimes known as the Busacker-Gowen algorithm.
• MATCHING can be thought of as a flow optimization problem

• We take the links in the graph to represent flow capacities:
  • Let $n$ be the number of boys (girls) and $m$ the number of possible matches.
  • all links have capacity 1;
  • an perfect matching corresponds to a flow from source to sink with value $n$, where $n$ is the number of boys/girls.

• But we have just seen how to solve this problem on $O(n(n + m))$ time!
Outline

Matching

Flow networks

Third-year projects

The stable marriage problem
• The task:
  • Students choose projects, and each project is associated with one or more supervisors;
  • supervisors have maximum loads;
  • projects have maximum numbers of participants;
  • The Lab Manager must assign students to projects so as to maximize student satisfaction.

• The general set-up of (unranked) choices, projects and supervisors can still be modelled as a flow network:
To take account of rankings, we introduce costs into the flow network:

- choice rankings (for Student 1 only) are shown in blue;
- we model these rankings as costs per unit of flow.

Here, an optimal assignment of students to projects corresponds to a least cost maximal flow in the network.
So, we have a mathematical model of our problem:
- the structure of students, projects, supervisors, choices, loadings ... is represented as a flow network with costs;
- an optimal assignment of $m$ students corresponds to a least cost integral flow of size $m$

Now use the Busacker-Gowen algorithm to solve this problem.

Remember:
- this algorithm runs incrementally, gradually increasing the flow from the source to the sink by one unit on each pass;
- after $m$ passes, a least-cost flow of size $m$ has been found;
- at the $(m + 1)$st pass, the current least-cost flow of size $m$ is revised to yield a least-cost flow of size $m + 1$;
- it runs in polynomial time.
• How the program is used:
  • students input their choices (by a certain deadline) using a web form;
  • once the deadline is passed the Third Year Project Administrator runs the assignment software;
  • the assignment software computes the optimum assignment, asks for verification, and writes the assignment to the CS administration databases;
  • the assignment is then published on the internet.

• Important considerations:
  • all students should be treated fairly (everyone equally likely to get his first choice);
  • all supervisors should be treated fairly (loads should be as even as possible).
• Program structure

- User input
- Data entry webpage
- Student choices
- ACSO database
- Randomizer
- Flow network
- Busacker-Gowen algorithm
- Assignments
- VDU/file output

• And here is a run ...
p:\programs\java\...\ProjAssignV4>java ProjAssign

Safe flag is set to true
maxNumberOfStudentsAssigned= 20

<table>
<thead>
<tr>
<th>Number</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
<th>5th</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>18</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>4</td>
<td>9</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>98</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
<td>8</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>108</td>
</tr>
</tbody>
</table>
Outline

Matching

Flow networks

Third-year projects

The stable marriage problem
<table>
<thead>
<tr>
<th>Matching</th>
<th>Flow networks</th>
<th>Third-year projects</th>
<th>The stable marriage problem</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Arnold Schwarzenegger
Arnold Schwarzenegger
Matching

Arnold Schwarzenegger

Flow networks

Ingrid Bergman

Third-year projects

Al Roth

The stable marriage problem

Lloyd Shapley
The stable matching problem

- Consider the following (slightly idealized) matchmaking problem.
- We are given:
  - a set of $n$ boys and $n$ girls;
  - a strict ranking, for each boy, of all the girls
  - a strict ranking, for each girl, of all the boys
- We want to compute:
  - a 1–1 pairing of boys with girls in which, for every boy $a$ and girl $b$, either $a$ prefers his partner to $b$ or $b$ prefers her partner to $a$. (Such a pairing is called a stable matching.)
- It helps to draw the arrangement as a complete bipartite graph:

- It is not in the least obvious that a stable matching always exists.
• It helps to draw the arrangement as a complete bipartite graph:

![Bipartite Graph]

• It is not in the least obvious that a stable matching always exists.
The Gale-Shapley algorithm generates a matching as follows

begin Gale-Shapley(Boys’ rankings, Girls’ rankings)
until all boys are engaged do
for each boy \( a \) with no fiancée do
    \( a \) proposes to the girl he most prefers among those that \( a \) has not yet proposed to
for each girl \( b \) with new proposals do
    let \( a \) be \( b \)’s most preferred new suitor
    if \( b \) has no fiancé
        \( b \) gets engaged to \( a \)
    if \( b \) prefers \( a \) to her existing fiancé
        \( b \) cancels her existing engagement
        \( b \) gets engaged to \( a \)
All the engaged couples get married
the end
Theorem

*The Gale-Shapley algorithm terminates with a stable matching.*

Proof.

The algorithm terminates
Theorem

The Gale-Shapley algorithm terminates with a stable matching.

Proof.
The algorithm terminates

Once a girl receives a proposal, she always has some fiancé or other.
If there are any unengaged girls, some boy will propose to one of them eventually.
Hence all the girls get engaged
Hence all the boys get engaged and the algorithm stops.
Theorem

The Gale-Shapley algorithm terminates with a stable matching.

Proof.

The algorithm terminates

The resulting matching is stable
Theorem

The Gale-Shapley algorithm terminates with a stable matching.

Proof.
The algorithm terminates

The resulting matching is stable

Suppose \( \langle a, b \rangle \) and \( \langle a', b' \rangle \), are distinct married couples produced by the algorithm. If \( b \) prefers \( a' \) to \( a \), then \( a' \) never proposed to \( b \). But \( a' \) proposed to all girls better (for him) than or equal to \( b' \). Therefore, \( a' \) does not prefer \( b \) to \( b' \). \qed
- Exercise: show that Gale-Shapley terminates in time $O(n^2)$, where $n$ is the number of boys (or girls).
Theorem

The stable matching, $M$, produced by the Gale-Shapley algorithm is optimal for boys: if boy $a$ is married to girl $b$ in $M$, but prefers girl $b'$, then there is no stable matching $M'$ in which $a$ is married to $b'$.

Proof.

If $a$ prefers $b'$ to $b$, then the pair $\langle a, b' \rangle \in M'$ must have been a rejected proposal or cancelled engagement in the construction of $M$. 
Theorem

The stable matching, $M$, produced by the Gale-Shapley algorithm is optimal for boys: if boy $a$ is married to girl $b$ in $M$, but prefers girl $b'$, then there is no stable matching $M'$ in which $a$ is married to $b'$.

Proof.

If $a$ prefers $b'$ to $b$, then the pair $\langle a, b' \rangle \in M'$ must have been a rejected proposal or cancelled engagement in the construction of $M$. 
Proof.

Let $\langle a, b' \rangle \in M'$ be the first pair rejected or cancelled in the construction of $M$.

Let $a''$ be such that $b'$ rejects/cancels $a$ for $a''$ in construction of $M$. So $b'$ prefers $a''$ to $a$.

Let $b''$ be the partner of $a''$ in $M'$ (stable). So $a''$ prefers $b''$ to $b'$. But then the pair $\langle a'', b'' \rangle$ must have been rejected/cancelled in construction of $M$, contradicting fact that $\langle a, b' \rangle$ is first. $\blacksquare$
Proof.

Let \( \langle a, b' \rangle \in M' \) be the first pair rejected or cancelled in the construction of \( M \).

Let \( a'' \) be such that \( b' \) rejects/cancels \( a \) for \( a'' \) in construction of \( M \). So \( b' \) prefers \( a'' \) to \( a \).

Let \( b'' \) be the partner of \( a'' \) in \( M' \) (stable). So \( a'' \) prefers \( b'' \) to \( b' \).

But then the pair \( \langle a'', b'' \rangle \) must have been rejected/cancelled in construction of \( M \), contradicting fact that \( \langle a, b' \rangle \) is first.
Summary

• In this lecture, we have considered:
  • a pseudo-polynomial-time algorithm for determining optimal flows in flow networks;
  • the reduction of maximum matching to flow optimization (with bounded capacities);
  • a pseudo-polynomial-time algorithm for determining minimal cost optimal flows in flow networks with costs;
  • a practical application of this algorithm;
  • an algorithm for producing stable matchings.

Items marked in blue are not examinable.

• In several cases, we found that analysing these (practically useful) algorithms yields interesting mathematical results.

• Reading:
  • G+T Ch 16.