CSE 101
Algorithm Design and Analysis
Sanjoy Dasgupta, Ragesh Jaiswal Russell Impagliazzo
mej016@eng.ucsd.edu
russell@cs.ucsd.edu
Lecture 20 (optional lecture): Approximation
OPTIMIZATION PROBLEMS

• In general, when you try to solve a problem, you are trying to find the best solution from among a large space of possibilities.
• Some optimization problems are hard, unless \( P=NP \)
• We still need to solve them
• Relax our notion of \(``solve''\)': instead of finding a solution \( GS \) so that \( \text{Value}(GS) \geq \text{Value}(OS) \) for every other solution, \( OS \)

Just guarantee that \( GS \) is approximately optimal with approximation ratio \( C : \text{Value}(GS) \geq \text{Value}(OS)/C \)

\( (C=1: \text{exact optimal, larger } C \text{ is worse}) \)
3. What is an algorithm you could use to select the *best* option if you can’t select 2 cookies from the same row or column?

Also known as max weight bipartite perfect matching, Actually a real problem that does come up.
3. What is an algorithm you could use to select the best option if you can’t select 2 cookies from the same row or column?

Pick largest element in matrix
Remove its row and column
Recurse on remaining n-1 by n-1 matrix
<table>
<thead>
<tr>
<th></th>
<th>56</th>
<th>76</th>
<th>69</th>
<th>60</th>
<th>75</th>
<th>51</th>
</tr>
</thead>
<tbody>
<tr>
<td>61</td>
<td>77</td>
<td>74</td>
<td>72</td>
<td>80</td>
<td>58</td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>97</td>
<td>94</td>
<td>88</td>
<td>99</td>
<td>92</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>68</td>
<td>59</td>
<td>52</td>
<td>65</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>81</td>
<td>79</td>
<td>71</td>
<td>85</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>67</td>
<td>73</td>
<td>57</td>
<td>70</td>
<td>46</td>
<td></td>
</tr>
</tbody>
</table>

3. What is an algorithm you could use to select the best option if you can’t select 2 cookies from the same row or column?
3. What is an algorithm you could use to select the best option if you can’t select 2 cookies from the same row or column?
3. What is an algorithm you could use to select the best option if you can’t select 2 cookies from the same row or column?

\[
\begin{align*}
99+81+74+60+50+40 &= 404 \\
99+81+72+69+47+46 &= 414 \\
92+78+75+73+72+68 &= 458
\end{align*}
\]

So our greedy algorithm isn’t optimal
3. What is an algorithm you could use to select the best option if you can’t select 2 cookies from the same row or column?

\[
\begin{align*}
99+81+74+60+50+40 &= 404 \\
99+81+72+69+47+46 &= 414 \\
92+78+75+73+72+68 &= 458 \text{!!!!!!!}
\end{align*}
\]
IMMEDIATE BENEFIT VS OPPORTUNITY COSTS

IMMEDIATE BENEFIT: How does the choice we’re making now contribute to the objective function?

OPPORTUNITY COST: How does the choice we’re making now restrict future choices?

Greedy (usually) Take best immediate benefit and ignore opportunity costs.

Greedy is optimal: Best immediate benefits outweigh opportunity costs.

Greedy is good approximation: Immediate benefit is at least a large fraction of opportunity costs.
**Immediate benefit** ≥\(1/2\) * Opportunity cost

**Opportunity costs**

- \(82, 97, 94, 88, 92\)
- \(92 + 85\)

**Constants**

- \(99\)
- \(75\)
Theorem: Let GS be the greedy solution to MWBPM. Let OS be any set of positions that includes at most one per row and one per column. Then Total(GS) ≥ ½ Total(OS)

Greedy solution: Take largest matrix element. Remove its row and its column. Repeat for remaining elements.

Other solution: All we know is that it is a set of matrix elements, with at most one in each row and at most one in each column.
As things get removed

Step 1. 2 3. 4. 5. 6...... N

GS.  \( g_1 \).  \( g_2 \).  \( g_3 \) ................. \( g_N \)

OS.  \( o_1, o_2 \)  \( o_3, o_4 \)  ........

In each step, at most 2 elements of OS get removed: one in \( g_i \)'s column, one in \( g_i \)'s row. Both of these elements are at most \( g_i \), since \( g_i \) is largest remaining. By step N, all elements have been removed.
Value(OS) = \sum_{o \in OS} o = \sum_T (\text{total value of elements removed at step } T) \\
\leq \sum_T 2 \cdot g_T = 2 \sum_T g_T = 2 \text{Value} (GS)

So no other solution can be more than twice the value of GS

So GS has value at least \( \frac{1}{2} \) the optimum value.

Approximation ratio=2
Salesperson needs to visit every location and return to start, wants to minimize total distance travelled. Famous NP-complete problem.

Can assume distances obey triangle inequality:

\[ D(u,v) \leq D(u,w) + D(w,v) \]
Find the minimum spanning tree $T$

Do a DF traversal of $T$
A to E to B to F to C to F to D to F to B to E to A:

Short circuit to only include first appearance of each city: A to E to B to F to C to D to A
Let TSP be the cost of the minimum travelling salesperson tour
MST be the cost of the minimum spanning tree

Any tour connects the graph, so $TSP \geq MST$

In other words, MST is a lower bound on the cost. If we can show our algorithm has cost within some factor of this bound, the approximation ratio for our algorithm is that factor.
Find the minimum spanning tree $T$: $\text{Cost} = \text{MST}$

Do a DF traversal of $T$
A to E to B to F to C to F to D to F to B to E to A: We use each edge of the tree twice, Once in each direction. $\text{Cost} = 2\times \text{MST}$

Short circuit to only include first appearance of each city: A to E to B to F to C to D to A: By triangle inequality, this only makes cost smaller
Cost of our solution is at most 2*MST

Cost of any tour is at least MST.

Therefore, the cost of our solution is at most twice the optimal.

This algorithm had an approximation ratio of 2.

Best approximation ratio known: 1.5 by Christofides-Serdyukov, 1976
Many jobs requiring time $t_1, \ldots t_n$ need to be divided between two identical machines. We want to complete the jobs as early as possible, so want to minimize the greater of the two total times of jobs assigned to the two machines.

For example, if the times are 11, 7, 6, 5, 3, 4
We could give one machine 11, 3 and 4
The other 6,5, and 7
Both would finish in 18 time steps
GREEDY ALGORITHM

- Sort the jobs from longest to shortest.
- In order, assign each job to the machine with less load
- 11, 7, 6, 5, 4, 3 = 36
- M1: 11, 5, 3 = 19. 11, 4, 3 = 18
- M2: 7, 6, 4 = 17. 7, 6, 5 = 18
Use lower bounds on OPT, the optimal finish time
What is a lower bound on OPT?
Use lower bounds on OPT, the optimal finish time
What is a lower bound on OPT?
Total = $\sum t_i$
Max = $\max t_i$
OPT $\geq \frac{1}{2}$ Total
OPT $\geq$ Max $= t_1$
Theorem: GS ≤ 2OPT:

Proof: Let’s say that $T_1$ is the total time on $M_1$ and $T_2$ is the total time on $M_2$. Without loss of generality let’s say that $T_1 \geq T_2$. And let’s say that job $j$ was that last job added to $M_1$. 
Theorem: $\text{Cost(GS)} \leq 2\text{OPT}$:

In fact, this is true for any solution, since $\text{Cost(any solution)} \leq \text{Total} \leq 2*\text{OPT}$

So let’s try to do better
Let $T_1$ be the larger loaded side of the greedy algorithm, and $T_2$ the smaller side. Then $T_1 + T_2 = Total$ and $T_2 \leq T_1$, so $T_2 \leq \frac{1}{2} Total \leq OPT$

Let $t_j$, the j’th largest job, be the last job added to $T_1$ ‘s side. Before it was added, that side had the smaller load, so $T_1 - t_j \leq \frac{1}{2} (Total - t_j)$

$Cost(GS) = T_1 \leq \frac{1}{2} (Total + t_j) \leq OPT + \frac{1}{2} t_j \leq OPT + \frac{1}{2} Max \leq \frac{3}{2} OPT$

Because there are j jobs at least as large as $t_j$, $Total \geq j \cdot t_j$, 
Let $t_j$, the j’th largest job, be the last job added to $T_1$ ‘s side. Before it was added, that side had the smaller load, so $T_1 - t_j \leq \frac{1}{2} (Total - t_j)$

If $j = 1$, Cost(GS) = $t_1 = Max = OPT$

$j = 2$ is impossible

Cost(GS) = $T_1 \leq \frac{1}{2} (Total + t_j) \leq OPT + \frac{1}{2} t_j$

Because there are j jobs at least as large as $t_j$, Total $\geq j t_j$, so $\frac{1}{2} t_j \leq \frac{1}{2j} Total \leq \frac{1}{j} OPT$

Cost(GS) $\leq \left(1 + \frac{1}{j}\right) OPT, j \geq 3, so Cost(GS) \leq \frac{4}{3} OPT$
What happens if \( j = 3 \)?

- \( t_1 \) on one side, \( t_2 + t_3 \) on the other, then more jobs go on the first, but \( t_2 + t_3 \) stays the largest.

- Of the three largest jobs, \( t_1, t_2, t_3 \), two must go on the same side in any schedule. Therefore, \( \text{OPT} \) is at least \( t_2 + t_3 = \text{GS} \) in this case.

- If \( j \) is at least 4, the previous argument gives \( \text{OPT} \) is at most \( \text{GS} \leq \text{OPT} + \frac{1}{j} \text{OPT} = \frac{5}{4} \text{OPT} \).
- approximation ratio is at least $\frac{7}{6}$.

- $3,3,2,2,2$

- $\text{OPT } 3+3, 2+2+2$

- $\text{GS } 3+2+2=7, 3+2=5$

- But we’ll see a different algorithm using dynamic programming that can achieve any approximation ratio bigger than 1.
Sometimes it’s better to have a fast algorithm that gives a reasonable solution than a slow algorithm that finds the best solution.

Approximation algorithms can be quite simple, and try to find a balance between extremes. We make choices that are relatively safe, rather than the ones that are most likely to lead to an optimal solution.

Greedy algorithms are often good approximation algorithms, even when they aren’t optimal.