CPSC 420 Lecture 20: Today's announcements:

- HW3 is on Gradescope, due Mar 9, 23:59
- Examlet 3 on Mar 17 in class. Closed book & no notes
- Reading: Approximation Algorithms [Intro to Algs 4th Ed. by Cormen, Leiserson, Rivest, Stein Ch. 35]

Today's Plan

- NP-hardness
  - Hamiltonian cycle (and TSP)
- Approximation algorithms

Traveling Salesperson Problem (TSP)

Given graph G with positive weights on the edges and a number k, does G contain a Hamiltonian cycle with total edge weight $\leq k$?

Claim: TSP is NP-complete.

A. If we can solve TSP in polytime then we can solve HamCycle in polytime.

B. Given a sequence of vertices, we can check in polytime:

1. the sequence forms a cycle in G
2. the cycle visits all the vertices in G
3. the sum of the edges in the cycle is $\leq k$

Approximate Solutions for NP-hard Optimization Problems

An **optimization problem** asks for a maximum (or minimum) value solution to a problem.

For example,

MaxClique finds a maximum size clique in a given graph G.

MinVertexCover finds a minimum size vertex cover in a given graph G.

MinTSP finds a minimum weight TSP in a given edge-weighted graph G.

These problems are all NP-hard (but not NP-complete) so fast algorithms are unlikely.

What do we do?

Approximation Algorithms

An algorithm A is a $\rho(n)$-**approximation algorithm** if for every input I of size n with optimal solution value $\mathrm{OPT}(I)$,

$$\max\left\{\underbrace{\frac{\text{value } A(I)}{\mathrm{OPT}(I)}}_{\substack{\text{minimizing}\\\text{problems}}}, \underbrace{\frac{\mathrm{OPT}(I)}{\text{value } A(I)}}_{\substack{\text{maximizing}\\\text{problems}}}\right\} \le \rho(n).$$

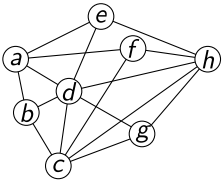
For example, for MinVertexCover, we want an algorithm A so that for all inputs I,

$$\frac{\text{value } A(I)}{\mathrm{OPT}(I)} \le \rho(n).$$

2-Approximation Algorithm for MinVertexCover

MatchVC

1. $S = \{\}$
2. Repeat
3. Pick arbitrary edge (u, v) in G
4. Remove u and v and their edges from G
5. Add u and v to S
6. Until G contains no edges

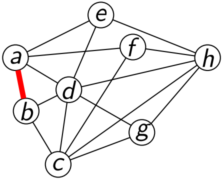


$S = \{\}$

2-Approximation Algorithm for MinVertexCover

MatchVC

1. $S = \{\}$
2. Repeat
3. Pick arbitrary edge (u, v) in G
4. Remove u and v and their edges from G
5. Add u and v to S
6. Until G contains no edges

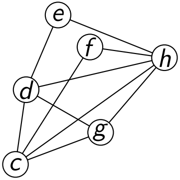


$S = \{\}$

2-Approximation Algorithm for MinVertexCover

MatchVC

1. $S = \{\}$
2. Repeat
3. Pick arbitrary edge (u, v) in G
4. Remove u and v and their edges from G
5. Add u and v to S
6. Until G contains no edges

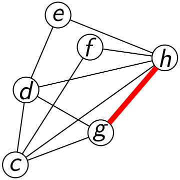


$S = \{a, b\}$

2-Approximation Algorithm for MinVertexCover

MatchVC

1. $S = \{\}$
2. Repeat
3. Pick arbitrary edge (u, v) in G
4. Remove u and v and their edges from G
5. Add u and v to S
6. Until G contains no edges

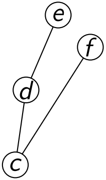


$S = \{a, b\}$

2-Approximation Algorithm for MinVertexCover

MatchVC

1. $S = \{\}$
2. Repeat
3. Pick arbitrary edge (u, v) in G
4. Remove u and v and their edges from G
5. Add u and v to S
6. Until G contains no edges

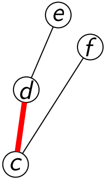


$S = \{a, b, g, h\}$

2-Approximation Algorithm for MinVertexCover

MatchVC

1. $S = \{\}$
2. Repeat
3. Pick arbitrary edge (u, v) in G
4. Remove u and v and their edges from G
5. Add u and v to S
6. Until G contains no edges

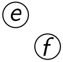


$S = \{a, b, g, h\}$

2-Approximation Algorithm for MinVertexCover

MatchVC

```
S = {}
Repeat
      Pick arbitrary edge (u, v) in G
      Remove u and v and their edges from G
      Add u and v to S
Until G contains no edges
```

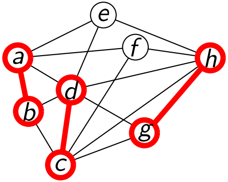


$S = \{a, b, g, h, c, d\}$

2-Approximation Algorithm for MinVertexCover

MatchVC

1. $S = \{\}$
2. Repeat
3. Pick arbitrary edge (u, v) in G
4. Remove u and v and their edges from G
5. Add u and v to S
6. Until G contains no edges



$S = \{a, b, g, h, c, d\}$

2-Approximation Algorithm for MinVertexCover

Claim: MatchVC is a 2-approx algorithm for MinVertexCover
Proof:

1. $\text{OPT}(G) \geq \#$ edges picked by $\text{MatchVC}(G)$
2. value $\text{MatchVC}(G) = 2 \times \#$ edges picked by $\text{MatchVC}(G)$

$\Rightarrow$

$$\frac{\text{value MatchVC}(G)}{\text{OPT}(G)} \leq 2$$

Why is 1. true?

2-Approximation Algorithm for MinVertexCover

Claim: MatchVC is a 2-approx algorithm for MinVertexCover
Proof:

1. $\mathrm{OPT}(G) \geq \#$ edges picked by $\mathrm{MatchVC}(G)$
2. value $\mathrm{MatchVC}(G) = 2 \times \#$ edges picked by $\mathrm{MatchVC}(G)$

$\Rightarrow$

$$\frac{\text{value MatchVC}(G)}{\mathrm{OPT}(G)} \leq 2$$

Why is 1. true?

Each picked edge must be covered by **any** vertex cover (including $\mathrm{OPT}(G)$ solution), and no two picked edges share an endpoint.

2-Approximation Algorithm for MinVertexCover

Claim: MatchVC is a 2-approx algorithm for MinVertexCover
Proof:

1. $\mathrm{OPT}(G) \geq \#$ edges picked by $\mathrm{MatchVC}(G)$
2. value $\mathrm{MatchVC}(G) = 2 \times \#$ edges picked by $\mathrm{MatchVC}(G)$

$\Rightarrow$

$$\frac{\text{value } \mathrm{MatchVC}(G)}{\mathrm{OPT}(G)} \leq 2$$

Why is 1. true?

Each picked edge must be covered by **any** vertex cover (including $\mathrm{OPT}(G)$ solution), and no two picked edges share an endpoint.

Note: We don't know $\mathrm{OPT}(G)$ but we can lower bound it.

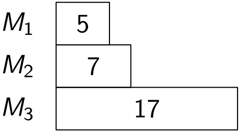
List Scheduling

Given a set of n jobs where job i must run uninterrupted for p_i time units, and m identical machines each of which can work on one job at a time. Find schedule of jobs on machines that minimizes the completion time (time when last job finishes).

GreedyLS: Whenever a machine becomes idle, assign next job to that machine.

$p = [\cancel{5}, \cancel{7}, \cancel{17}, 10, 9, 30]$

$m = 3$



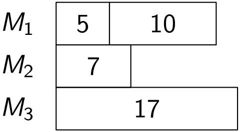
List Scheduling

Given a set of n jobs where job i must run uninterrupted for p_i time units, and m identical machines each of which can work on one job at a time. Find schedule of jobs on machines that minimizes the completion time (time when last job finishes).

GreedyLS: Whenever a machine becomes idle, assign next job to that machine.

$p = [\cancel{5}, \cancel{7}, \cancel{17}, \cancel{10}, 9, 30]$

$m = 3$



List Scheduling

Given a set of n jobs where job i must run uninterrupted for p_i time units, and m identical machines each of which can work on one job at a time. Find schedule of jobs on machines that minimizes the completion time (time when last job finishes).

GreedyLS: Whenever a machine becomes idle, assign next job to that machine.

$p = [\cancel{5}, \cancel{7}, \cancel{17}, \cancel{10}, \cancel{9}, 30]$

$m = 3$

M_1	5	10
M_2	7	9
M_3	17	

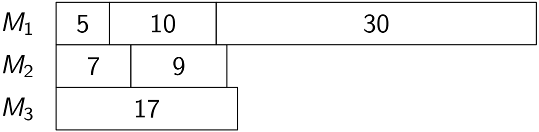
List Scheduling

Given a set of n jobs where job i must run uninterrupted for p_i time units, and m identical machines each of which can work on one job at a time. Find schedule of jobs on machines that minimizes the completion time (time when last job finishes).

GreedyLS: Whenever a machine becomes idle, assign next job to that machine.

$p = [\cancel{5}, \cancel{7}, \cancel{17}, \cancel{10}, \cancel{9}, \cancel{30}]$

$m = 3$



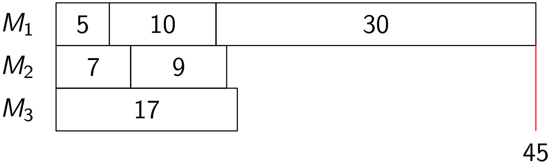
List Scheduling

Given a set of n jobs where job i must run uninterrupted for p_i time units, and m identical machines each of which can work on one job at a time. Find schedule of jobs on machines that minimizes the completion time (time when last job finishes).

GreedyLS: Whenever a machine becomes idle, assign next job to that machine.

$p = [\cancel{5}, \cancel{7}, \cancel{17}, \cancel{10}, \cancel{9}, \cancel{30}]$

$m = 3$



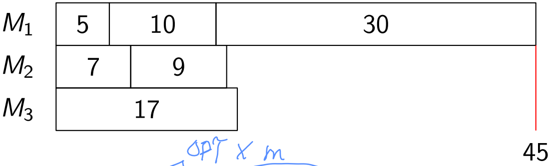
List Scheduling

Given a set of n jobs where job i must run uninterrupted for p_i time units, and m identical machines each of which can work on one job at a time. Find schedule of jobs on machines that minimizes the completion time (time when last job finishes).

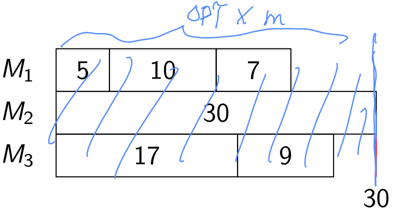
GreedyLS: Whenever a machine becomes idle, assign next job to that machine.

$p = [\cancel{5}, \cancel{7}, \cancel{17}, \cancel{10}, \cancel{9}, \cancel{30}]$

$m = 3$



Optimal

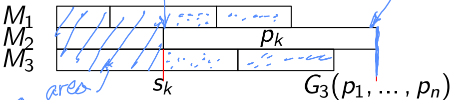


GreedyLS is a $(2 - \frac{1}{m})$-Approximation Alg. [Graham '66]

Let $G_m(p_1, \ldots, p_n)$ be completion time of GreedyLS schedule
Let $\mathrm{OPT}_m(p_1, \ldots, p_n)$ be minimum completion time

Claim: $G_m(p_1, \ldots, p_n) \leq (2 - \frac{1}{m})\mathrm{OPT}(p_1, \ldots, p_n)$
Proof: Let k be the last job to finish for GreedyLS. Let s_k be its start time.

start of last finishing job



area $= m \cdot s_k$

1. $s_k \leq \frac{1}{m} \sum_{i \neq k} p_i$ [all machines work nonstop before s_k in GreedyLS]

LB on OPT

2. $\mathrm{OPT}_m(p_1, \ldots, p_n) \geq p_k$ and $\mathrm{OPT}_m(p_1, \ldots, p_n) \geq \frac{1}{m} \sum_{i=1}^{n} p_i$
3. $G_m(p_1, \ldots, p_n) = s_k + p_k \leq \frac{1}{m} \sum_{i \neq k} p_i + p_k$ all but $(1 - 1/m)\, p_k$

$$= \frac{1}{m} \sum_{i=1}^{n} p_i + (1 - \frac{1}{m}) p_k \leq (2 - \frac{1}{m})\mathrm{OPT}(p_1, \ldots, p_n)$$

$\leq \mathrm{OPT}$ OPT

Sorting Job Sizes

Claim: GreedyLS is a $(\frac{3}{2} - \frac{1}{2m})$-approximation algorithm if $p_1 \geq p_2 \geq \cdots \geq p_n$

Proof: If $n \leq m$ then $G_m(p_1, \ldots, p_n) = \mathrm{OPT}(p_1, \ldots, p_n)$.

If $n > m$ then $\mathrm{OPT}(p_1, \ldots, p_n) \geq 2p_{m+1}$
because two jobs from biggest $m+1$ jobs must run on the same machine in any schedule $\Rightarrow$ completion time $\geq 2p_{m+1}$.

$p_k \leq p_{m+1}$ since GreedyLS schedules $p_1, \ldots, p_m$ first.

given that $p_1 \ldots p_m$ do not determine OPT

$$\text{As before } G_m(p_1, \ldots, p_n) = s_k + p_k \leq \frac{1}{m}\sum_{i=1}^{n} p_i + (1 - \frac{1}{m})p_k$$

$$\leq \mathrm{OPT}(p_1, \ldots, p_n) + (1 - \frac{1}{m})\frac{\mathrm{OPT}(p_1, \ldots, p_n)}{2}$$

$$= (\frac{3}{2} - \frac{1}{2m})\mathrm{OPT}(p_1, \ldots, p_n)$$