# CSC373

# Approximation Algorithms

# NP-Completeness

- NP-complete problems
  - Unlikely to have polynomial time algorithms to solve them
  - What do we do?

- One idea: approximation
  - Instead of solving them exactly, solve them approximately
  - Sometimes, we might want to use an approximation algorithm even when we can compute an exact solution in polynomial time (WHY?)

# Approximation Algorithms

- Decision versus optimization problems
  - Decision variant: "Does there exist a solution with objective $\geq k$?"
    - E.g., "Is there an assignment which satisfies at least $k$ clauses of a given CNF formula $\varphi$?"
  - Optimization variant: "Find a solution maximizing objective"
    - E.g., "Find an assignment which satisfies the maximum possible number of clauses of a given CNF formula $\varphi$."
  - If a decision problem is hard, then its optimization version is hard too
  - We'll focus on optimization variants

# Approximation Algorithms

- Objectives
  - Maximize (e.g., “profit”) or minimize (e.g., “cost”)

- Given problem instance $I$:
  - $ALG(I)$ = solution returned by our algorithm
  - $OPT(I)$ = some optimal solution
  - Approximation ratio of $ALG$ on instance $I$ is

$$\frac{profit\big(OPT(I)\big)}{profit\big(ALG(I)\big)} \quad \text{or} \quad \frac{cost\big(ALG(I)\big)}{cost\big(OPT(I)\big)}$$

  - Convention: approximation ratio $\geq 1$
    - “2-approximation” = half the optimal profit / twice the optimal cost

# Approximation Algorithms

- Worst-case approximation ratio
  - Worst approximation ratio across all possible problem instances $I$
  - $ALG$ has worst-case $c$-approximation if for **each problem instance** $I$…

$$profit\big(ALG(I)\big) \geq \frac{1}{c} \cdot profit\big(OPT(I)\big) \ or$$

$$cost\big(ALG(I)\big) \leq c \cdot cost\big(OPT(I)\big)$$

  - By default, we will always refer to approximation ratios in the worst case
  - Note: In some textbooks, you might see the approximation ratio flipped (e.g., $0.5$-approximation instead of 2-approximation)

# PTAS and FPTAS

- Arbitrarily close to 1 approximations
- PTAS: Polynomial time approximation scheme
  - For every $\epsilon > 0$, there is a $(1 + \epsilon)$-approximation algorithm that runs in time $poly(n)$ on instances of size $n$
    - Note: Could have exponential dependence on $1/\epsilon$
- FPTAS: Fully polynomial time approximation scheme
  - For every $\epsilon > 0$, there is a $(1 + \epsilon)$-approximation algorithm that runs in time $poly(n, 1/\epsilon)$ on instances of size $n$

# Approximation Landscape

Impossibility of better approximations assuming widely held beliefs like P $\neq$ NP

$n$ = parameter of problem at hand

- An FPTAS
  - E.g., the knapsack problem
- A PTAS but no FPTAS
  - E.g., the makespan problem (we'll see)
- $c$-approximation for a constant $c > 1$ but no PTAS
  - E.g., vertex cover and JISP (we'll see)
- $\Theta(\log n)$-approximation but no constant approximation
  - E.g., set cover
- No $n^{1-\epsilon}$-approximation for any $\epsilon > 0$
  - E.g., graph coloring and maximum independent set

# Approximation Techniques

- Greedy algorithms
  - Make decision on one element at a time in a greedy fashion without considering future decisions
- LP relaxation
  - Formulate the problem as an integer linear program (ILP)
  - “Relax” it to an LP by allowing variables to take real values
  - Find an optimal solution of the LP, “round” it to a feasible solution of the original ILP, and prove its approximate optimality
- Local search
  - Start with an arbitrary solution
  - Keep making “local” adjustments to improve the objective

# Greedy Approximation

# Makespan Minimization

# Makespan

- Problem
  - Input: $m$ identical machines, $n$ jobs, job $j$ requires processing time $t_j$
  - Output: Assign jobs to machines to minimize makespan

- Let $S[i]$ = set of jobs assigned to machine $i$ in a solution
- Constraints:
  - Each job must run contiguously on one machine
  - Each machine can process at most one job at a time
- Load on machine $i$: $L_i = \sum_{j \in S[i]} t_j$
- Goal: minimize the *maximum* load, i.e., makespan $L = \max_i L_i$

# Makespan

- Even the special case of $m = 2$ machines is already NP-hard by reduction from PARTITION

- PARTITION
  - Input: Set $S$ containing $n$ integers
  - Question: Does there exist a partition of $S$ into two sets with equal sum? (A partition of $S$ into $S_1, S_2$ means $S_1 \cap S_2 = \emptyset$ and $S_1 \cup S_2 = S$)

- Exercise!
  - Show that PARTITION is NP-complete by reduction from SUBSET-SUM
  - Show that Makespan with $m = 2$ is NP-hard by reduction from PARTITION

# Makespan

- Greedy list-scheduling algorithm
  - Consider the $n$ jobs in some "nice" sorted order
  - Assign each job $j$ to a machine with the smallest load so far

- Note: Implementable in $O(n \log m)$ using priority queue

- Back to greedy…?
  - But this time, we can't hope that greedy will be optimal
  - We can still hope that it is approximately optimal

- Which order?

# Makespan

- Theorem [Graham 1966]
  - Regardless of the order, greedy gives a 2-approximation.
  - This was one of the first worst-case approximation analyses

- Let optimal makespan = $L^*$

- To show that makespan under the greedy solution is not much worse than $L^*$, we need to show that $L^*$ cannot be too low

# Makespan

- Theorem [Graham 1966]
  - Regardless of the order, greedy gives a 2-approximation.
- Fact 1: $L^* \geq \max_j t_j$
  - Some machine must process job with highest processing time
- Fact 2: $L^* \geq \frac{1}{m} \sum_j t_j$
  - Total processing time is $\sum_j t_j$
  - At least one machine must do at least $1/m$ of this work (the pigeonhole principle)

# Makespan

- Theorem [Graham 1966]
  - Regardless of the order, greedy gives a 2-approximation.

- Proof:
  - Suppose machine $i$ is the bottleneck under greedy (so $L = L_i$)
  - Let $j^*$ be the last job scheduled on machine $i$ by greedy
  - Right before $j^*$ was assigned to $i$, $i$ had the smallest load
    - Load of the other machines could have only increased from then
    - $L_i - t_{j^*} \leq L_k, \forall k$
  - Average over all $k$ : $L_i - t_{j^*} \leq \frac{1}{m} \sum_j t_j$
  - $L_i \leq t_{j^*} + \frac{1}{m} \sum_j t_j \leq L^* + L^* = 2L^*$

Fact 1

Fact 2

# Makespan

- Theorem [Graham 1966]
  - Regardless of the order, greedy gives a 2-approximation.

- Is our analysis tight?
  - Essentially.
  - By averaging over $k \neq i$ in the previous slide, one can show a slightly better $2 - 1/m$ approximation
  - There is an example where greedy has approximation as bad as $2 - 1/m$
  - So, $2 - 1/m$ is exactly tight.

# Makespan

- Tight example:
  - $m(m-1)$ jobs of length $1$, followed by one job of length $m$
  - Greedy evenly distributes unit length jobs on all $m$ machines, and assigning the last heavy job makes makespan $m-1+m=2m-1$
  - Optimal makespan is $m$ by evenly distributing unit length jobs among $m-1$ machines and putting the single heavy job on the remaining

- Idea:
  - It seems keeping heavy jobs at the end is bad.
  - So, let's just start with them first!

# Makespan Revisited

- Greedy LPT (Longest Processing Time First)
  - Run the greedy algorithm but consider jobs in a non-increasing order of their processing time
  - Suppose $t_1 \geq t_2 \geq \cdots \geq t_n$
- Fact 3: If the bottleneck machine $i$ has only one job $j$, then the solution is optimal
  - Current solution has $L = L_i = t_j$
  - We know $L^* \geq t_j$ from Fact 1
- Fact 4: If there are more than $m$ jobs, then $L^* \geq 2 \cdot t_{m+1}$
  - The first $m + 1$ jobs each have processing time at least $t_{m+1}$
  - By the pigeonhole principle, the optimal solution must put at least two of them on the same machine

# Makespan Revisited

- Theorem
  - Greedy LPT achieves ${}^{3}/_{2}$-approximation
- Proof:
  - Similar to the proof for arbitrary ordering
  - Consider a bottleneck machine $i$ and the job $j^*$ that was last scheduled on this machine by the greedy algorithm
  - Case 1: Machine $i$ has only one job $j^*$
    - By Fact 3, greedy is optimal in this case (i.e. 1-approximation)

# Makespan Revisited

- Theorem
  - Greedy LPT achieves $3/2$-approximation

- Proof:
  - Similar to the proof for arbitrary ordering
  - Consider a bottleneck machine $i$ and the job $j^*$ that was last scheduled on this machine by the greedy algorithm
  - Case 2: Machine $i$ has at least two jobs
    - Job $j^*$ must have $t_{j^*} \leq t_{m+1}$
    - As before, $L = L_i = \underbrace{\left(L_i - t_{j^*}\right)}_{\leq L^*} + \underbrace{t_{j^*}}_{\leq L^*/2} \leq 1.5\, L^*$

Same as before

$t_{j^*} \leq t_{m+1}$ and Fact 4

# Makespan Revisited

- Theorem
  - Greedy LPT achieves $3/2$-approximation
  - Is our analysis tight? No!

- Theorem [Graham 1966]
  - Greedy LPT achieves $\left(\frac{4}{3} - \frac{1}{3\mathrm{m}}\right)$-approximation
  - Is Graham's approximation tight?
    - Yes.
    - In the upcoming example, greedy LPT is as bad as $\frac{4}{3} - \frac{1}{3m}$

# Makespan Revisited

- Tight example for Greedy LPT:
  - 2 jobs each of lengths $m, m+1, \ldots, 2m-1$
  - One more job of length $m$
  - Greedy-LPT has makespan $4m-1$ (verify!)
  - OPT has makespan $3m$ (verify!)
  - Thus, approximation ratio is at least as bad as $\frac{4m-1}{3m} = \frac{4}{3} - \frac{1}{3m}$

# Unweighted Vertex Cover

# Unweighted Vertex Cover

- Problem
  - Input: Undirected graph $G = (V, E)$
  - Output: Vertex cover $S$ of minimum cardinality

  - Recall: $S$ is vertex cover if every edge has at least one of its two endpoints in $S$
  - We already saw that this problem is NP-hard

- Q: What would be a good greedy algorithm for this problem?

# Unweighted Vertex Cover

- Greedy edge-selection algorithm:
  - Start with $S = \emptyset$
  - While there exists an edge whose both endpoints are not in $S$, add *both* its endpoints to $S$

- Hmm…
  - Why are we selecting edges rather than vertices?
  - Why are we adding both endpoints?
  - We'll see..

# Unweighted Vertex Cover

GREEDY-VERTEX-COVER($G$)

---

$S \leftarrow \varnothing$.

$E' \leftarrow E$.

WHILE ($E' \neq \varnothing$)

  Let $(u, v) \in E'$ be an arbitrary edge.

  $M \leftarrow M \cup \{(u, v)\}$. ⟵ $M$ is a matching

  $S \leftarrow S \cup \{u\} \cup \{v\}$. ⟵ every vertex cover must take at least one of these; we take both

  Delete from $E'$ all edges incident to either $u$ or $v$.

RETURN $S$.

# Unweighted Vertex Cover

- Theorem:
  - Greedy edge-selection algorithm for unweighted vertex cover achieves 2-approximation.
- Observation 1:
  - For any vertex cover $S^*$ and any matching $M$, $|S^*| \geq |M|$, where $|M|$ = number of edges in $M$
  - Proof: $S^*$ must contain at least one endpoint of each edge in $M$
- Observation 2:
  - Greedy algorithm finds a vertex cover of size $|S| = 2 \cdot |M|$
- Hence, $|S| \leq 2 \cdot |S^*|$, where $S^*$ = min vertex cover

# Unweighted Vertex Cover

- Corollary:
  - If $M^*$ is a *maximum* matching, and $M$ is a *maximal* matching, then $|M| \geq \frac{1}{2}|M^*|$
- Proof:
  - By design, $|M| = \frac{1}{2}|S|$
  - $|S| \geq |M^*|$ (Observation 1)
  - Hence, $|M| \geq \frac{1}{2}|M^*|$ ■

- This greedy algorithm is also a 2-approximation to the problem of finding a maximum cardinality matching
  - However, the max cardinality matching problem can be solved exactly in polynomial time using a more complex algorithm

# Unweighted Vertex Cover

- What about a greedy vertex selection algorithm?
  - Start with $S = \emptyset$
  - While $S$ is not a vertex cover:
    - Choose a vertex $v$ which maximizes the number of uncovered edges incident on it
    - Add $v$ to $S$
  - Gives $O(\log d_{\max})$ approximation, where $d_{\max}$ is the maximum degree of any vertex
    - But unlike the edge-selection version, this generalizes to set cover
    - For set cover, $O(\log d_{\max})$ approximation ratio is the best possible in polynomial time unless P=NP

# Unweighted Vertex Cover

NOT IN SYLLABUS

- Theorem [Dinur-Safra 2004]:
  - Unless P = NP, there is no polynomial-time $\rho$-approximation algorithm for unweighted vertex cover for any constant $\rho < 1.3606$.

On the Hardness of Approximating Minimum Vertex Cover

Irit Dinur* Samuel Safra†

May 26, 2004

**Abstract**

We prove the Minimum Vertex Cover problem to be NP-hard to approximate to within a factor of 1.3606, extending on previous PCP and hardness of approximation technique. To that end, one needs to develop a new proof framework, and borrow and extend ideas from several fields.

# Unweighted Vertex Cover

NOT IN SYLLABUS

- Theorem [Khot-Regev 2008]:
  - Unless the "unique games conjecture" is violated, there is no polynomial-time $\rho$-approximation algorithm for unweighted vertex cover for any constant $\rho < 2$.

**Vertex Cover Might be Hard to Approximate to within $2 - \varepsilon$**

Subhash Khot * Oded Regev †

**Abstract**

Based on a conjecture regarding the power of unique 2-prover-1-round games presented in [Khot02], we show that vertex cover is hard to approximate within any constant factor better than 2. We actually show a stronger result, namely, based on the same conjecture, vertex cover on $k$-uniform hypergraphs is hard to approximate within any constant factor better than $k$.

# Unweighted Vertex Cover

NOT IN SYLLABUS

- How does one prove a lower bound on the approximation ratio of any polynomial-time algorithm?
  - We prove that if there is a polynomial-time $\rho$-approximation algorithm for the problem with $\rho <$ some bound, then some widely believed conjecture is violated
  - For example, we can prove that given a polynomial time $\rho$-approximation algorithm to vertex cover for any constant $\rho < 1.3606$, we can use this algorithm as a subroutine to solve the 3SAT decision problem in polynomial time, implying P=NP
  - Similar technique can be used to reduce from other widely believed conjectures, which may give different (sometimes better) bounds
  - Beyond the scope of this course

# Weighted Vertex Cover

# Weighted Vertex Cover

- Problem
  - Input: Undirected graph $G = (V, E)$, weights $w : V \to R_{\geq 0}$
  - Output: Vertex cover $S$ of minimum total weight

- The same greedy algorithm doesn't work
  - Gives arbitrarily bad approximation
  - Obvious modifications which try to take weights into account also don't work
  - Need another strategy…

# LP Relaxation

# ILP Formulation

- For each vertex $v$, create a binary variable $x_v \in \{0,1\}$ indicating whether vertex $v$ is chosen in the vertex cover
- Then, computing min weight vertex cover is equivalent to solving the following integer linear program

$$
\begin{aligned}
&\min \Sigma_v\, w_v \cdot x_v \\
&\text{subject to} \\
&x_u + x_v \geq 1, \qquad \forall (u,v) \in E \\
&x_v \in \{0,1\}, \qquad \forall v \in V
\end{aligned}
$$

# LP Relaxation

- What if we solve the "LP relaxation" of the original ILP?
  - Just convert all integer variables to real variables

**ILP with binary variables**

$\min \Sigma_v w_v \cdot x_v$

subject to

$x_u + x_v \geq 1, \qquad \forall (u, v) \in E$

$x_v \in \{0,1\}, \qquad \forall v \in V$

**LP with real variables**

$\min \Sigma_v w_v \cdot x_v$

subject to

$x_u + x_v \geq 1, \qquad \forall (u, v) \in E$

$x_v \geq 0, \qquad \forall v \in V$

# Rounding LP Solution

- What if we solve the "LP relaxation" of the original ILP?
  - Let's say we are minimizing objective $c^T x$
  - Since the LP minimizes this over a larger feasible space than the ILP, optimal LP objective value $\leq$ optimal ILP objective value
  - Let $x_{LP}^*$ be an optimal LP solution (which we can compute efficiently) and $x_{ILP}^*$ be an optimal ILP solution (which we can't compute efficiently)
    - $c^T x_{LP}^* \leq c^T x_{ILP}^*$
    - But $x_{LP}^*$ may have non-integer values
    - Efficiently round $x_{LP}^*$ to an ILP feasible solution $\hat{x}$ without increasing the objective too much
    - If we prove $c^T \hat{x} \leq \rho \cdot c^T x_{LP}^*$, then we will also have $c^T \hat{x} \leq \rho \cdot c^T x_{ILP}^*$
    - Thus, our algorithm will achieve $\rho$-approximation

# Rounding LP Solution

- What if we solve the "LP relaxation" of the original ILP?
  - If we are <u>maximizing</u> $c^T x$ instead of minimizing, then it's reversed:
    - Optimal LP objective value $\geq$ optimal ILP objective value, i.e., $c^T x^*_{LP} \geq c^T x^*_{ILP}$
    - Efficiently round $x^*_{LP}$ to an ILP feasible solution $\hat{x}$ without <u>decreasing</u> the objective too much
    - If we prove $c^T \hat{x} \geq (^1/_\rho) \cdot c^T x^*_{LP}$, then $c^T \hat{x} \geq (^1/_\rho) \cdot c^T x^*_{ILP}$
    - Thus, our algorithm will achieve $\rho$-approximation

# Weighted Vertex Cover

- Consider LP optimal solution $x^*$
  - Let $\hat{x}_v = 1$ whenever $x_v^* \geq 0.5$ and $\hat{x}_v = 0$ otherwise

  - Claim 1: $\hat{x}$ is a feasible solution of ILP (i.e., a vertex cover)
    - For every edge $(u, v) \in E$, at least one of $\{x_u^*, x_v^*\}$ is at least $0.5$
    - So at least one of $\{\hat{x}_u, \hat{x}_v\}$ is 1 ■

**ILP with binary variables**

$$\min \Sigma_v w_v \cdot x_v$$
subject to
$$x_u + x_v \geq 1, \qquad \forall (u, v) \in E$$
$$x_v \in \{0,1\}, \qquad \forall v \in V$$

**LP with real variables**

$$\min \Sigma_v w_v \cdot x_v$$
subject to
$$x_u + x_v \geq 1, \qquad \forall (u, v) \in E$$
$$x_v \geq 0, \qquad \forall v \in V$$

# Rounding LP Solution

- Consider LP optimal solution $x^*$
  - Let $\hat{x}_v = 1$ whenever $x_v^* \geq 0.5$ and $\hat{x}_v = 0$ otherwise
  - Claim 2: $\sum_v w_v \cdot \hat{x}_v \leq 2 * \sum_v w_v \cdot x_v^*$
    - Weight only increases when some $x_v^* \in [0.5,1]$ is rounded *up* to 1
    - At most doubling the variable, so at most doubling the weight ■

**ILP with binary variables**

$$\min \Sigma_v w_v \cdot x_v$$
subject to
$$x_u + x_v \geq 1, \qquad \forall (u,v) \in E$$
$$x_v \in \{0,1\}, \qquad \forall v \in V$$

**LP with real variables**

$$\min \Sigma_v w_v \cdot x_v$$
subject to
$$x_u + x_v \geq 1, \qquad \forall (u,v) \in E$$
$$x_v \geq 0, \qquad \forall v \in V$$

# Rounding LP Solution

- Consider LP optimal solution $x^*$
  - Let $\hat{x}_v = 1$ whenever $x_v^* \geq 0.5$ and $\hat{x}_v = 0$ otherwise
  - Hence, $\hat{x}$ is a vertex cover with weight at most $2 *$ LP optimal value $\leq 2 *$ ILP optimal value

**ILP with binary variables**

$$\begin{aligned} &\min \Sigma_v w_v \cdot x_v \\ &\text{subject to} \\ &x_u + x_v \geq 1, \qquad \forall (u,v) \in E \\ &x_v \in \{0,1\}, \qquad \forall v \in V \end{aligned}$$

**LP with real variables**

$$\begin{aligned} &\min \Sigma_v w_v \cdot x_v \\ &\text{subject to} \\ &x_u + x_v \geq 1, \qquad \forall (u,v) \in E \\ &x_v \geq 0, \qquad \forall v \in V \end{aligned}$$

# General LP Relaxation Strategy

- Your NP-complete problem amounts to solving
  - Max $c^T x$ subject to $Ax \leq b, x \in \mathbb{N}$ (need not be binary)

- Instead, solve:
  - Max $c^T x$ subject to $Ax \leq b, x \in \mathbb{R}_{\geq 0}$ (LP relaxation)
    - LP optimal value $\geq$ ILP optimal value (for maximization)
  - $x^*$ = LP optimal solution
  - Round $x^*$ to $\hat{x}$ such that $c^T \hat{x} \geq \frac{c^T x^*}{\rho} \geq \frac{\text{ILP optimal value}}{\rho}$
  - Gives $\rho$-approximation
    - Info: Best $\rho$ you can hope to get via this approach for a particular LP-ILP combination is called the *integrality gap*

# Local Search Paradigm

# Local Search

- Heuristic paradigm
  - Sometimes it might provably return an optimal solution
  - But even if not, it might give a good approximation

- Template
  - Start with some initial feasible solution $S$
  - While there is a "better" solution $S'$ in the **local neighborhood** of $S$
  - Switch to $S'$

- Need to define:
  - Which initial feasible solution should we start from?
  - What is "better"?
  - What is "local neighborhood"?

# Local Search

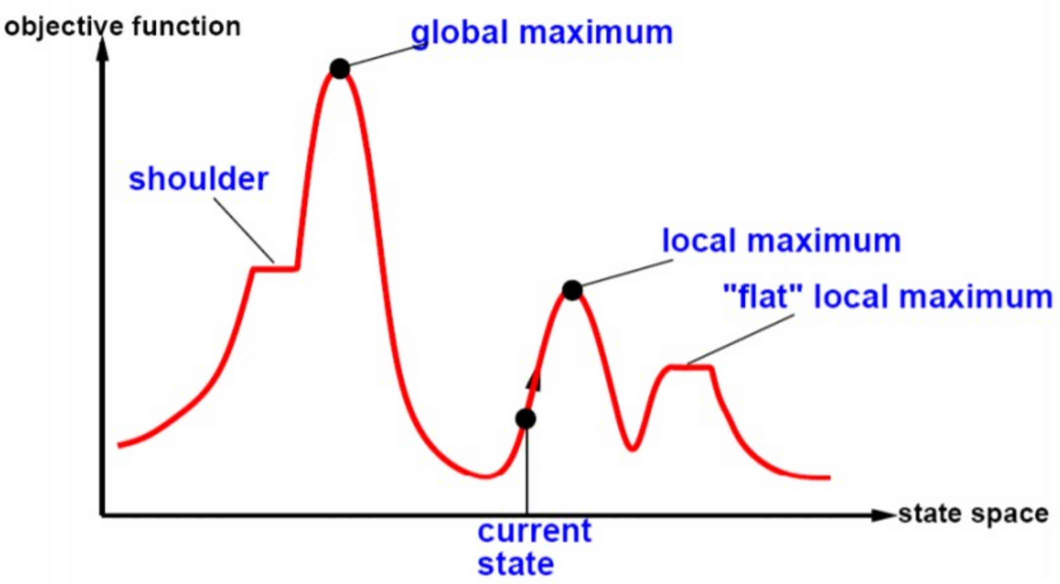
- For some problems, local search provably returns an optimal solution

- Example: network flow
  - Initial solution: zero flow
  - Local neighborhood: all flows that can be obtained by augmenting the current flow along a path in the residual graph
  - Better: Higher flow value

- Example: LP via simplex
  - Initial solution: a vertex of the polytope
  - Local neighborhood: neighboring vertices
  - Better: better objective value

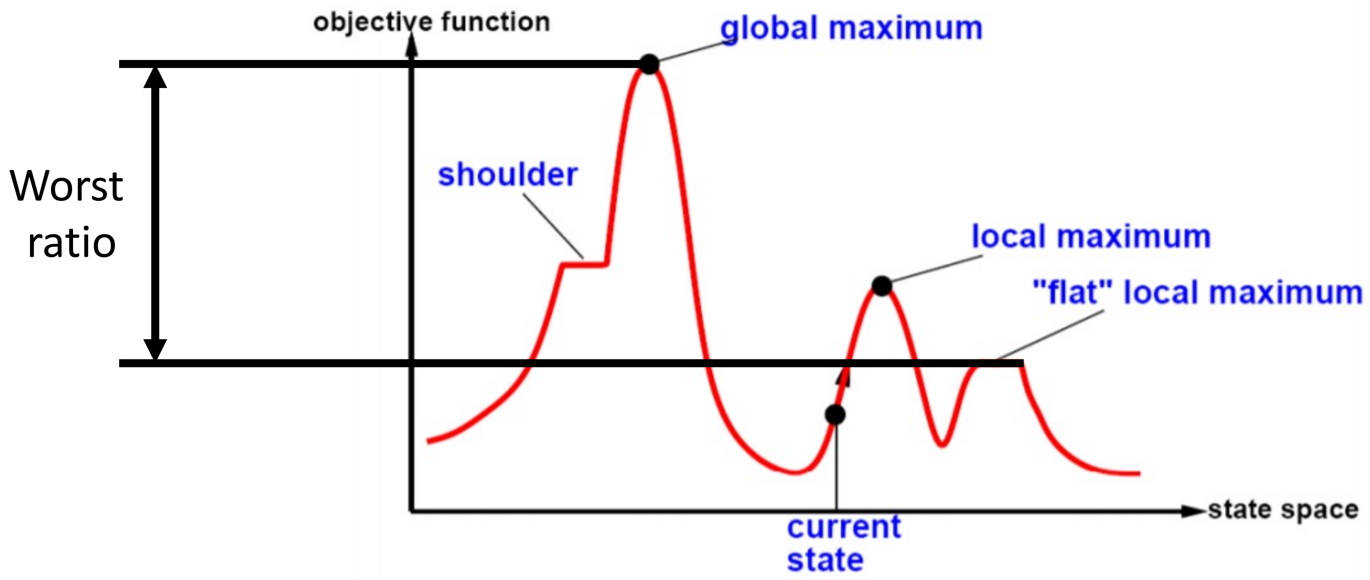
# Local Search

- But sometimes it doesn’t return an optimal solution, and “gets stuck” in a local maxima



# Local Search

- In that case, we want to bound the worst-case ratio between the global optimum and the worst local optimum (the worst solution that local search might return)



# Max-Cut

# Max-Cut

- Problem
  - Input: An undirected graph $G = (V, E)$
  - Output: A partition $(A, B)$ of $V$ that maximizes the number of edges going across the cut, i.e., maximizes $|E'|$ where $E' = \{(u, v) \in E \mid u \in A, v \in B\}$

- This is also known to be an NP-hard problem

- What is a natural local search algorithm for this problem?
  - Given a current partition, what small change can you do to improve the objective value?

# Max-Cut

- Local Search
  - Initialize $(A, B)$ arbitrarily.
  - While there is a vertex $u$ such that moving $u$ to the other side improves the objective value:
    - Move $u$ to the other side.

- When does moving $u$, say from $A$ to $B$, improve the objective value?
  - When $u$ has more incident edges going within the cut than across the cut, i.e., when $|\{(u, v) \in E \mid v \in A\}| > |\{(u, v) \in E \mid v \in B\}|$

# Max-Cut

- Local Search
  - Initialize $(A, B)$ arbitrarily.
  - While there is a vertex $u$ such that moving $u$ to the other side improves the objective value:
    - Move $u$ to the other side.

- Why does the algorithm stop?
  - Every iteration increases the number of edges across the cut by at least $1$, so the algorithm must stop in at most $|E|$ iterations

# Max-Cut

- Local Search
  - Initialize $(A, B)$ arbitrarily.
  - While there is a vertex $u$ such that moving $u$ to the other side improves the objective value:
    - Move $u$ to the other side.

- Approximation ratio?
  - At the end, every vertex has at least as many edges going across the cut as within the cut
  - Hence, at least half of all edges must be going across the cut
    - Exercise: Prove this formally by writing equations.

# Weighted Max-Cut

NOT IN SYLLABUS

- Variant
  - Now we're given integral edge weights $w: E \to \mathbb{N}$
  - The goal is to maximize the total *weight* of edges going across the cut

- Algorithm
  - The same algorithm works...
  - But we move $u$ to the other side if the total *weight* of its incident edges going within the cut is greater than the total *weight* of its incident edges going across the cut

# Weighted Max-Cut

NOT IN SYLLABUS

- Number of iterations?
  - Unweighted case: #edges going across the cut must increase by at least $1$, so it takes at most $|E|$ iterations
  - Weighted case: total *weight* of edges going across the cut must increase by at least $1$, but this could take up to $\sum_{e \in E} w_e$ iterations, which can be *exponential* in the input length
    - There are examples where the local search actually takes exponentially many steps
    - Fun exercise: Design an example where the number of iterations is exponential in the input length.

# Weighted Max-Cut

NOT IN SYLLABUS

- Number of iterations?
  - But we can find a $2 + \epsilon$ approximation in time polynomial in the input length and $\frac{1}{\epsilon}$
  - The idea is to only move vertices when it "sufficiently improves" the objective value

# Weighted Max-Cut

NOT IN SYLLABUS

- Better approximations?
  - Theorem [Goemans-Williamson 1995]:

    There exists a polynomial time algorithm for max-cut with approximation ratio $\frac{2}{\pi} \cdot \min_{0 \le \theta \le \pi} \frac{\theta}{1-\cos\theta} \approx 0.878$

    - Uses "semidefinite programming" and "randomized rounding"
    - Note: The literature from here on uses approximation ratios $\le 1$, so we will follow that convention in the remaining slides.

  - Assuming the "unique games conjecture", this approximation ratio is tight

# Exact Max-$k$-SAT

# Exact Max-$k$-SAT

- Problem
  - Input: An exact $k$-SAT formula $\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, where each clause $C_i$ has exactly $k$ literals, and a weight $w_i \geq 0$ of each clause $C_i$
  - Output: A truth assignment $\tau$ maximizing the total weight of clauses satisfied under $\tau$

- Let us denote by $W(\tau)$ the total weight of clauses satisfied under $\tau$

- What is a good definition of "local neighborhood"?

# Exact Max-$k$-SAT

- Local neighborhood:
  - $N_d(\tau)$ = set of all truth assignments $\tau'$ which differ from $\tau$ in the values of at most $d$ variables

- Theorem: The local search with $d = 1$ gives a $^2/_3$ approximation to Exact Max-2-SAT.

# Exact Max-$k$-SAT

NOT IN SYLLABUS

- **Theorem:** The local search with $d = 1$ gives a ${}^{2}/_{3}$ approximation to Exact Max-2-SAT.
- **Proof:**
  - Let $\tau$ be a local optimum
    - $S_0$ = set of clauses not satisfied under $\tau$
    - $S_1$ = set of clauses from which exactly one literal is true under $\tau$
    - $S_2$ = set of clauses from which both literals are true under $\tau$
    - $W(S_0), W(S_1), W(S_2)$ be the corresponding total weights

    - Goal: $W(S_1) + W(S_2) \geq {}^{2}/_{3} \cdot \big(W(S_0) + W(S_1) + W(S_2)\big)$
      - Equivalently, $W(S_0) \leq {}^{1}/_{3} \cdot \big(W(S_0) + W(S_1) + W(S_2)\big)$

# Exact Max-$k$-SAT

NOT IN SYLLABUS

- Theorem: The local search with $d = 1$ gives a ${}^{2}/_{3}$ approximation to Exact Max-2-SAT.
- Proof:
  - We say that clause $C$ "involves" variable $j$ if it contains $x_j$ or $\overline{x_j}$
  - $A_j$ = set of clauses in $S_0$ involving variable $j$
    - Let $W(A_j)$ be the total weight of such clauses
  - $B_j$ = set of clauses in $S_1$ involving variable $j$ such that it is the literal of variable $j$ that is true under $\tau$
    - Let $W(B_j)$ be the total weight of such clauses

# Exact Max-$k$-SAT

NOT IN SYLLABUS

- Theorem: The local search with $d = 1$ gives a ${}^{2}/_{3}$ approximation to Exact Max-2-SAT.
- Proof:
  - $2\, W(S_0) = \sum_j W(A_j)$
    - Every clause in $S_0$ is counted twice on the RHS
  - $W(S_1) = \sum_j W(B_j)$
    - Every clause in $S_1$ is only counted once on the RHS for the variable whose literal was true under $\tau$
  - For each $j : W(A_j) \leq W(B_j)$
    - From local optimality of $\tau$, since otherwise flipping the truth value of variable $j$ would have increased the total weight

# Exact Max-$k$-SAT

NOT IN SYLLABUS

- Theorem: The local search with $d = 1$ gives a ${}^{2}/{}_{3}$ approximation to Exact Max-2-SAT.
- Proof:
  - $2\, W(S_0) \leq W(S_1)$
    - Summing the third equation on the last slide over all $j$, and then using the first two equations on the last slide
  - Hence:
    - $3\, W(S_0) \leq W(S_0) + W(S_1) \leq W(S_0) + W(S_1) + W(S_2)$
    - Precisely the condition we wanted to prove…
    - QED!

# Exact Max-$k$-SAT

- Higher $d$?
  - Searches over a larger neighborhood
  - May get a better approximation ratio, but increases the running time as we now need to check if any neighbor in a large neighborhood provides a better objective

  - The bound is still $2/3$ for $d = o(n)$
  - For $d = \Omega(n)$, the neighborhood size is exponential
  - But the approximation ratio is…
    - At most $4/5$ with $d < {}^{n}/_{2}$
    - 1 (i.e. optimal solution is always reached) with $d = {}^{n}/_{2}$

# Exact Max-$k$-SAT

- Better approximation ratio?
  - We can learn something from our proof
  - Note that we did not use anything about $W(S_2)$, and simply added it at the end
  - If we could also guarantee that $W(S_0) \leq W(S_2)$...
    - Then we would get $4\, W(S_0) \leq W(S_0) + W(S_1) + W(S_2)$, which would give a $^3/_4$ approximation
  - Result (without proof):
    - This can be done by including just one more assignment in the neighborhood: $N(\tau) = N_1(\tau) \cup \{\tau^c\}$, where $\tau^c$ = complement of $\tau$

# Exact Max-$k$-SAT

- What if we do not want to modify the neighborhood?
  - A slightly different tweak also works
  - We want to weigh clauses in $W(S_2)$ more because when we get a clause through $S_2$, we get more robustness (it can withstand changes in single variables)

- Modified local search:
  - Start at arbitrary $\tau$
  - While there is an assignment in $N_1(\tau)$ that improves the potential $1.5\, W(S_1) + 2\, W(S_2)$
    - Switch to that assignment

# Exact Max-$k$-SAT

- Modified local search:
  - Start at arbitrary $\tau$
  - While there is an assignment in $N_1(\tau)$ that improves the potential $1.5\, W(S_1) + 2\, W(S_2)$
    - Switch to that assignment

- Note:
  - This is the first time that we're using a definition of "better" in local search paradigm that does not quite align with the ultimate objective we want to maximize
  - This is called "non-oblivious local search"

# Exact Max-$k$-SAT

- Modified local search:
  - Start at arbitrary $\tau$
  - While there is an assignment in $N_1(\tau)$ that improves the potential $1.5\, W(S_1) + 2\, W(S_2)$
    - Switch to that assignment

- Result (without proof):
  - Modified local search gives ${}^{3}/_{4}$-approximation to Exact Max-2-SAT

# Exact Max-$k$-SAT

- More generally:
  - The same technique works for higher values of $k$
  - Gives $\frac{2^k-1}{2^k}$ approximation for Exact Max-$k$-SAT
    - In the next lecture, we will achieve the same approximation ratio much more easily through a different technique

- Note: This ratio is $^7/_8$ for Exact Max-3-SAT
  - Theorem [Håstad]: Achieving $^7/_8 + \epsilon$ approximation where $\epsilon > 0$ is NP-hard.
    - Uses PCP (probabilistically checkable proofs) technique