CMPSCI 611: Advanced Algorithms

Lecture 25: More Approximation Algorithms and Review

Andrew McGregor

Last Compiled: February 1, 2024

Outline

Formulating Vertex Cover as a Linear (?) Program

- Given graph $G = (V, E)$, for each node $v \in V$, create variable x_v
- For each edge $(u, v) \in E$, create constraint $x_v + x_u \geq 1$

Formulating Vertex Cover as a Linear (?) Program

- Given graph $G = (V, E)$, for each node $v \in V$, create variable x_v
- For each edge $(u, v) \in E$, create constraint $x_v + x_u \geq 1$

Minimize $\sum_{v \in V} x_v$ subject to

$$\begin{aligned} x_v + x_u &\geq 1 && \text{for all } (u, v) \in E \\ x_v &\leq 1 && \text{for all } v \in V \\ x_v &\geq 0 && \text{for all } v \in V \end{aligned}$$

Does this mean we can solve Vertex Cover in poly-time?

Formulating Vertex Cover as a Linear (?) Program

- Given graph $G = (V, E)$, for each node $v \in V$, create variable x_v
- For each edge $(u, v) \in E$, create constraint $x_v + x_u \geq 1$

Minimize $\sum_{v \in V} x_v$ subject to

$$\begin{aligned} x_v + x_u &\geq 1 \quad \text{for all } (u,v) \in E \\ x_v &\leq 1 \quad \text{for all } v \in V \\ x_v &\geq 0 \quad \text{for all } v \in V \end{aligned}$$

Does this mean we can solve Vertex Cover in poly-time? No, need to constrain $x_v \in \{0, 1\}$. Program is an *integer* linear program (ILP).

Formulating Vertex Cover as a Linear (?) Program

- Given graph $G = (V, E)$, for each node $v \in V$, create variable x_v
- For each edge $(u, v) \in E$, create constraint $x_v + x_u \geq 1$

Minimize $\sum_{v \in V} x_v$ subject to

$$\begin{aligned} x_v + x_u &\geq 1 \quad \text{for all } (u, v) \in E \\ x_v &\leq 1 \quad \text{for all } v \in V \\ x_v &\geq 0 \quad \text{for all } v \in V \end{aligned}$$

Does this mean we can solve Vertex Cover in poly-time? No, need to constrain $x_v \in \{0, 1\}$. Program is an *integer* linear program (ILP).

Aside: When the graph is bipartite, something magical happens: the optimal solution will automatically be integral.

LP Relaxation

- Vertex cover can be expressed as the following integer program

LP Relaxation

- Vertex cover can be expressed as the following integer program
- Minimize $\sum_{v \in V} x_v$ subject to

$$
\begin{aligned}
x_v + x_u &\geq 1 \quad \text{for all } (u,v) \in E \\
x_v &\leq 1 \quad \text{for all } v \in V \\
x_v &\geq 0 \quad \text{for all } v \in V
\end{aligned}
$$

where each $x_v \in \{0,1\}$.

LP Relaxation

- Vertex cover can be expressed as the following integer program
- Minimize $\sum_{v \in V} x_v$ subject to

$$\begin{aligned} x_v + x_u &\geq 1 \quad \text{for all } (u,v) \in E \\ x_v &\leq 1 \quad \text{for all } v \in V \\ x_v &\geq 0 \quad \text{for all } v \in V \end{aligned}$$

  where each $x_v \in \{0, 1\}$.
- Relax: Ignore $x_v \in \{0, 1\}$ constraint.

LP Relaxation

- Vertex cover can be expressed as the following integer program
- Minimize $\sum_{v \in V} x_v$ subject to

$$\begin{aligned} x_v + x_u &\geq 1 \quad \text{for all } (u,v) \in E \\ x_v &\leq 1 \quad \text{for all } v \in V \\ x_v &\geq 0 \quad \text{for all } v \in V \end{aligned}$$

  where each $x_v \in \{0,1\}$.
- Relax: Ignore $x_v \in \{0,1\}$ constraint.
- Solve: Let $\hat{x}_v$ be optimal solution.

LP Relaxation

- Vertex cover can be expressed as the following integer program
- Minimize $\sum_{v \in V} x_v$ subject to

$$\begin{aligned} x_v + x_u &\geq 1 \quad \text{for all } (u,v) \in E \\ x_v &\leq 1 \quad \text{for all } v \in V \\ x_v &\geq 0 \quad \text{for all } v \in V \end{aligned}$$

  where each $x_v \in \{0,1\}$.
- Relax: Ignore $x_v \in \{0,1\}$ constraint.
- Solve: Let $\hat{x}_v$ be optimal solution.
- Round: Let $x'_v = 1$ if $\hat{x}_v \geq 1/2$ and 0 otherwise.

LP Relaxation

- Vertex cover can be expressed as the following integer program
- Minimize $\sum_{v \in V} x_v$ subject to

$$\begin{aligned} x_v + x_u &\geq 1 \quad \text{for all } (u,v) \in E \\ x_v &\leq 1 \quad \text{for all } v \in V \\ x_v &\geq 0 \quad \text{for all } v \in V \end{aligned}$$

  where each $x_v \in \{0, 1\}$.
- Relax: Ignore $x_v \in \{0, 1\}$ constraint.
- Solve: Let $\hat{x}_v$ be optimal solution.
- Round: Let $x'_v = 1$ if $\hat{x}_v \geq 1/2$ and 0 otherwise.
- Final solution is feasible for the original ILP and is a 2-approx.

LP Relaxation

- Vertex cover can be expressed as the following integer program
- Minimize $\sum_{v \in V} x_v$ subject to

$$\begin{aligned} x_v + x_u &\geq 1 \quad \text{for all } (u,v) \in E \\ x_v &\leq 1 \quad \text{for all } v \in V \\ x_v &\geq 0 \quad \text{for all } v \in V \end{aligned}$$

  where each $x_v \in \{0, 1\}$.
- Relax: Ignore $x_v \in \{0, 1\}$ constraint.
- Solve: Let $\hat{x}_v$ be optimal solution.
- Round: Let $x'_v = 1$ if $\hat{x}_v \geq 1/2$ and 0 otherwise.
- Final solution is feasible for the original ILP and is a 2-approx.
  - $\hat{x}_v + \hat{x}_u \geq 1$ implies $x'_v + x'_u \geq 1$ since at least one of $\hat{x}_v$ or $\hat{x}_u$ is $\geq 1/2$.

LP Relaxation

- Vertex cover can be expressed as the following integer program
- Minimize $\sum_{v \in V} x_v$ subject to

$$\begin{aligned} x_v + x_u &\geq 1 \quad \text{for all } (u, v) \in E \\ x_v &\leq 1 \quad \text{for all } v \in V \\ x_v &\geq 0 \quad \text{for all } v \in V \end{aligned}$$

  where each $x_v \in \{0, 1\}$.
- Relax: Ignore $x_v \in \{0, 1\}$ constraint.
- Solve: Let $\hat{x}_v$ be optimal solution.
- Round: Let $x'_v = 1$ if $\hat{x}_v \geq 1/2$ and 0 otherwise.
- Final solution is feasible for the original ILP and is a 2-approx.
  - $\hat{x}_v + \hat{x}_u \geq 1$ implies $x'_v + x'_u \geq 1$ since at least one of $\hat{x}_v$ or $\hat{x}_u$ is $\geq 1/2$.
  - After rounding, objective function at most doubles:

$$\sum_{v \in V} x'_v \leq 2 \sum_{v \in V} \hat{x}_v = 2\text{OPT}$$

Linear Programming: Review

Primal and Dual Linear Programs:

Primal LP	Dual LP
$\max \mathbf{c}^T\mathbf{x}$	$\min \mathbf{y}^T\mathbf{b}$
$\mathbf{A}\mathbf{x} \leq \mathbf{b}$	$\mathbf{y}^T\mathbf{A} \geq \mathbf{c}^T$
$\mathbf{x} \geq \mathbf{0}$	$\mathbf{y} \geq \mathbf{0}$

Theorem

Let $\mathrm{OPT}_{\text{primal}}$ be optimal solution of Primal LP and let $\mathrm{OPT}_{\text{dual}}$ be optimal solution of Dual LP: If both are bounded and feasible,

$$\mathrm{OPT}_{\text{primal}} = \mathrm{OPT}_{\text{dual}}$$

and hence, any feasible solution of the dual LP upper bounds $\mathrm{OPT}_{\text{primal}}$.

Linear Programming: Review

Primal and Dual Linear Programs:

Primal LP	Dual LP
$\max \mathbf{c}^T\mathbf{x}$	$\min \mathbf{y}^T\mathbf{b}$
$\mathbf{A}\mathbf{x} \leq \mathbf{b}$	$\mathbf{y}^T\mathbf{A} \geq \mathbf{c}^T$
$\mathbf{x} \geq \mathbf{0}$	$\mathbf{y} \geq \mathbf{0}$

Theorem

Let $\mathrm{OPT}_{\mathsf{primal}}$ *be optimal solution of Primal LP and let* $\mathrm{OPT}_{\mathsf{dual}}$ *be optimal solution of Dual LP: If both are bounded and feasible,*

$$\mathrm{OPT}_{\mathsf{primal}} = \mathrm{OPT}_{\mathsf{dual}}$$

and hence, any feasible solution of the dual LP upper bounds $\mathrm{OPT}_{\mathsf{primal}}$*.*

Applications of duality include a) max flow equals min cut and b) the max matching size equals the min vertex cover size in a bipartite graph.

LPs can be solved in poly-time but adding integral constraints makes the problem NP-hard.

Outline

Approximation Ratios

Definition

An algorithm for a minimization problem is an α-approximation if for all instances,

$$\frac{\text{value returned by the algorithm}}{\text{optimal value}} \leq \alpha \; .$$

For a maximization problem, we want the reciprocal to be at most α.

Examples:

- 2-approx for max-cut (local search technique)
- $3/2$-approx for metric traveling salesperson
- 2-approx for metric k-center clustering (in homework)
- $O(\log n)$-approx for weighted set-cover (charging technique)
- 2-approx for vertex cover (LP relaxation technique)
- $1 + \epsilon$-approx for generalized knapsack running in $O(n^3/\epsilon)$ time (via rounding the input values).

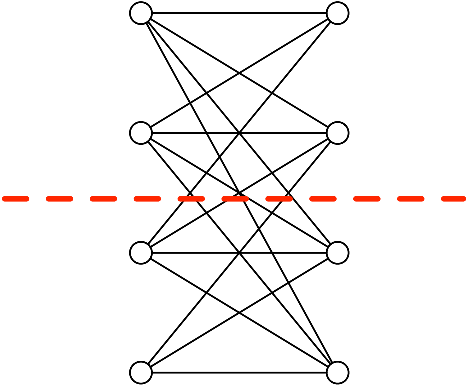
A reference of what approximation factors are known check out:

`http://www.csc.kth.se/~viggo/wwwcompendium/`

Tight Example

The following is an example where the local search algorithm for max-cut gets stuck at a 2-approximation.

- The max cut has size 16 but the cut indicated has size 8.
- The is no node where switching the side of the node strictly increases the size of the cut.



Outline

Linear Programs

Approximation Algorithms

Divide and Conquer

Greedy Algorithms

Dynamic Programming and Shortest Paths

Network Flows

Randomized Algorithms

NP Completeness

Outline

Divide and Conquer Methodology

- Goal: Solve problem P on an instance I of "size" n.
- Divide & Conquer Method:
  - Transform I into smaller instances $I_1, \ldots, I_a$ each of "size" n/b
  - Solve problem P on each of $I_1, \ldots, I_a$ by recursion
  - Combine the solutions to get a solution of I
- Examples: Merge Sort, Strassen's Algorithm, Minimum Distance, Fourier Transform.

Divide and Conquer Methodology

- Goal: Solve problem P on an instance I of "size" n.
- Divide & Conquer Method:
  - Transform I into smaller instances $I_1, \ldots, I_a$ each of "size" n/b
  - Solve problem P on each of $I_1, \ldots, I_a$ by recursion
  - Combine the solutions to get a solution of I
- Examples: Merge Sort, Strassen's Algorithm, Minimum Distance, Fourier Transform.

Let $T(n)$ be running time of algorithm on instance of size n. Then

$$T(1) = \Theta(1), T(n) = aT(n/b) + \Theta(n^\alpha)$$

where $\Theta(n^\alpha)$ is time to make new instances and combine solutions.

Theorem (Master Theorem)

If a, b, α are constants, then $T(n) = \begin{cases} \Theta(n^\alpha) & \text{if } \alpha > \log_b a \\ \Theta(n^{\log_b a}) & \text{if } \alpha < \log_b a \\ \Theta(n^\alpha \log n) & \text{if } \alpha = \log_b a \end{cases}$.

Cartoon


MATHEMATICIANS ARE WEIRD
YOU KNOW THAT THING THAT WAS 2.3728639?
YES?
WE GOT IT DOWN TO 2.3728596
Thunderous applause
smbc-comics.com

Outline

Generic Problem and Greedy Algorithms

Definition

A *subset system* $S = (E, \mathcal{I})$ is a finite set E with a collection $\mathcal{I}$ of subsets E such that:

$$\text{if } B \in \mathcal{I} \text{ and } A \subset B \text{ then } A \in \mathcal{I}$$

i.e., "$\mathcal{I}$ is closed under inclusion"

Problem Given a subset system $S = (E, \mathcal{I})$ and weight function $w : E \to \mathbb{R}^+$, find $A \in \mathcal{I}$ such that $w(A) = \sum_{e \in A} w(e)$ is maximized.

Algorithm (Greedy)

1. $A = \emptyset$
2. *Sort elements of E by non-increasing weight*
3. *For each $e \in E$: If $A + e \in \mathcal{I}$ then $A \leftarrow A + e$*

Matroid Definition and Theorem

Definition

A matroid is a subset system $(E, \mathcal{I})$ that satisfies the exchange property: if $A, B \in \mathcal{I}$ such that $|A| < |B|$, then $A + e \in \mathcal{I}$ for some $e \in B \setminus A$.

Theorem

For any subset system $(E, \mathcal{I})$, the greedy algorithm solves the optimization problem for $(E, \mathcal{I})$ if and only if $(E, \mathcal{I})$ is a matroid.

- A matroid can also be characterized by the cardinality theorem.
- Maximum bipartite matching can be expressed as intersection of two matroids and can therefore be solved in polynomial time.
- Solving the intersection of three matroids becomes NP-hard.

Outline

Dynamic Programming and Shortest Paths

When to use dynamic programming...

- *Optimal Substructure*: The solution to the problem can be found using solutions to smaller sub-problems.
- *Overlap of Sub-Problems*: By taking advantage of the fact that many identical sub-problems are created, a dynamic programming algorithm may be more efficient than a divide and conquer algorithm.

Shortest path algorithms...

- *Floyd-Warshall Algorithm*: $O(|V|^3)$
- *Dijkstra's Algorithm*: Positive weights! $O(|E| + |V| \log |V|)$.
- *Seidel's Algorithm*: Unweighted Graphs! $O(|V|^{2.38})$ running time.

Outline

Definitions

Input:

- Directed Graph $G = (V, E)$
- Capacities $C(u, v) > 0$ for $(u, v) \in E$ and $C(u, v) = 0$ for $(u, v) \notin E$
- A source node s, and sink node t

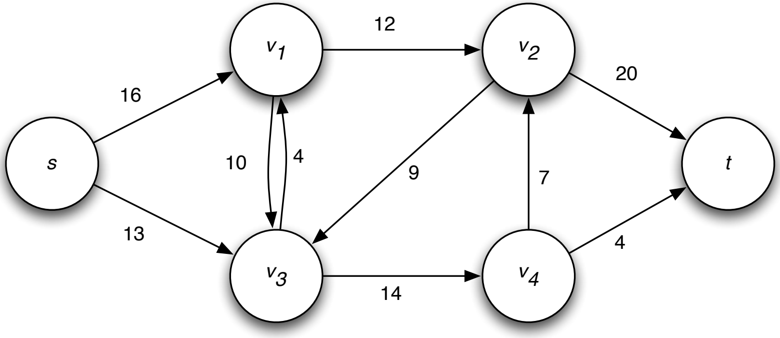
Output: A flow f from s to t where $f : V \times V \to \mathbb{R}$ satisfies

- Skew-symmetry: $\forall u, v \in V, f(u, v) = -f(v, u)$
- Conservation of Flow: $\forall v \in V - \{s, t\}, \sum_{u \in V} f(u, v) = 0$
- Capacity Constraints: $\forall u, v \in V,\ f(u, v) \leq C(u, v)$

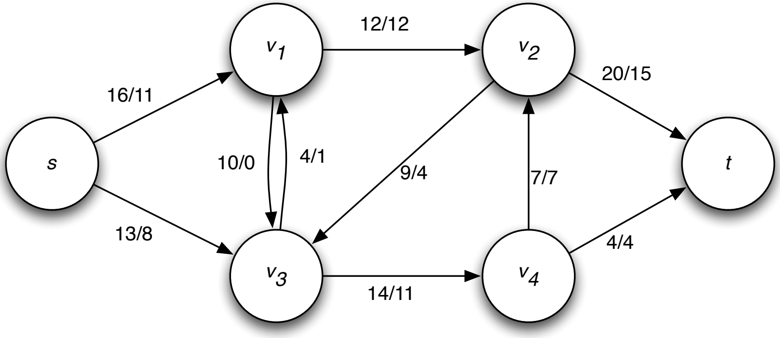
Goal: Maximize *"size of the flow"*, i.e., the total flow coming leaving s:

$$|f| = \sum_{v \in V} f(s, v)$$

Capacity



Capacity/Flow



Cut Definitions

Definition

An $s - t$ cut of G is a partition of the vertices into two sets A and B such that $s \in A$ and $t \in B$.

Definition

The capacity of a cut (A, B) is $C(A, B) = \sum_{u \in A, v \in B} C(u, v)$

Definition

The flow across a cut (A, B) is $f(A, B) = \sum_{u \in A, v \in B} f(u, v)$

Theorem (Max-Flow Min-Cut)

For any flow network and flow f, the following statements are equivalent:

1. *f is a maximum flow.*
2. *There exists an $s - t$ cut (A, B) such that $|f| = C(A, B)$*

Went over Ford-Fulkerson Algorithm with Edmonds-Karp Heuristic to find max-flow.

Outline

Probability and Examples

- For arbitrary events A and B,

$$\mathbb{P}[A \text{ and } B] = \mathbb{P}[A \text{ given } B]\,\mathbb{P}[B]$$

  and A and B are *independent* if $\mathbb{P}[A \text{ and } B] = \mathbb{P}[A]\,\mathbb{P}[B]$.
- Union Bound: $\mathbb{P}[A \text{ or } B] \leq \mathbb{P}[A] + \mathbb{P}[B]$
- Expectation: $\mathbb{E}[X] = \sum_r r\mathbb{P}[X = r]$
- Linearity of expectation: $\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$
- Variance random variable: $\mathbb{V}[X] = \sigma_X^2 = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right]$
- Linearity of variance if X and Y are independent:

$$\mathbb{V}[X + Y] = \mathbb{V}[X] + \mathbb{V}[Y]$$

Examples: Quicksort, Karger's Randomized Min-Cut Algorithm, Schwartz-Zippel, Lazy Select, Balls and Bins...

Tail Bounds

Theorem (Markov)

Let Y be a non-negative random variable. Then, for any $t > 0$,

$$\mathbb{P}\left[Y \geq tE(X)\right] \leq 1/t \ .$$

Theorem (Chebyshev)

Let X be any random variable. Then, for any $t > 0$,

$$\mathbb{P}\left[|X - E(X)| \geq t\right] \leq Var(X)/t^2 \ .$$

Theorem

Let $X_1, \ldots, X_n$ be independent boolean random variables and $X = \sum_i X_i$. Then for any $\delta > 0$,

$$\mathbb{P}\left[X > (1+\delta)\mu\right] < e^{-\delta^2\mu/3} \ \ \textit{and} \ \ \mathbb{P}\left[X < (1-\delta)\mu\right] < e^{-\delta^2\mu/2}$$

Outline

NP Completeness

1. Given decision problems Π and Π', then $\Pi \leq_p \Pi'$ means you can, in polynomial time, transform any instance I of Π into an instance $f(I)$ of Π' such that the answer for $f(I)$ is the same as the answer for I.

NP Completeness

1. Given decision problems Π and Π', then $\Pi \leq_p \Pi'$ means you can, in polynomial time, transform any instance I of Π into an instance $f(I)$ of Π' such that the answer for $f(I)$ is the same as the answer for I.
2. P: Problems with a poly-time algorithm

NP Completeness

1. Given decision problems Π and Π', then $\Pi \leq_p \Pi'$ means you can, in polynomial time, transform any instance I of Π into an instance $f(I)$ of Π' such that the answer for $f(I)$ is the same as the answer for I.
2. P: Problems with a poly-time algorithm
3. NP: Problems with a poly-time algorithm taking advice:

NP Completeness

1. Given decision problems Π and Π', then $\Pi \leq_p \Pi'$ means you can, in polynomial time, transform any instance I of Π into an instance $f(I)$ of Π' such that the answer for $f(I)$ is the same as the answer for I.
2. *P*: Problems with a poly-time algorithm
3. *NP*: Problems with a poly-time algorithm taking advice:
   - If the answer should be "yes", then there exists advice that leads the algorithm to output "yes"

NP Completeness

1. Given decision problems Π and Π', then $\Pi \leq_p \Pi'$ means you can, in polynomial time, transform any instance I of Π into an instance $f(I)$ of Π' such that the answer for $f(I)$ is the same as the answer for I.
2. *P*: Problems with a poly-time algorithm
3. *NP*: Problems with a poly-time algorithm taking advice:
   - If the answer should be "yes", then there exists advice that leads the algorithm to output "yes"
   - If the answer is "no", then there doesn't exist advice that would lead the algorithm to output "yes"

NP Completeness

1. Given decision problems Π and Π', then $\Pi \leq_p \Pi'$ means you can, in polynomial time, transform any instance I of Π into an instance $f(I)$ of Π' such that the answer for $f(I)$ is the same as the answer for I.
2. P: Problems with a poly-time algorithm
3. NP: Problems with a poly-time algorithm taking advice:
   - If the answer should be "yes", then there exists advice that leads the algorithm to output "yes"
   - If the answer is "no", then there doesn't exist advice that would lead the algorithm to output "yes"
4. A problem Π is NP-hard if for any $\Pi' \in NP$: $\Pi' \leq_P \Pi$

NP Completeness

1. Given decision problems Π and Π', then $\Pi \leq_p \Pi'$ means you can, in polynomial time, transform any instance I of Π into an instance $f(I)$ of Π' such that the answer for $f(I)$ is the same as the answer for I.
2. P: Problems with a poly-time algorithm
3. NP: Problems with a poly-time algorithm taking advice:
   - If the answer should be "yes", then there exists advice that leads the algorithm to output "yes"
   - If the answer is "no", then there doesn't exist advice that would lead the algorithm to output "yes"
4. A problem Π is NP-hard if for any $\Pi' \in NP$: $\Pi' \leq_P \Pi$
5. A problem Π is NP-complete if $\Pi \in NP$ and Π is NP-hard

Theorem

3-SAT, CLIQUE, VERTEX-COVER... are NP-Complete.

NP Completeness

1. Given decision problems Π and Π', then $\Pi \leq_p \Pi'$ means you can, in polynomial time, transform any instance I of Π into an instance $f(I)$ of Π' such that the answer for $f(I)$ is the same as the answer for I.
2. *P*: Problems with a poly-time algorithm
3. *NP*: Problems with a poly-time algorithm taking advice:
   - If the answer should be "yes", then there exists advice that leads the algorithm to output "yes"
   - If the answer is "no", then there doesn't exist advice that would lead the algorithm to output "yes"
4. A problem Π is NP-hard if for any $\Pi' \in NP$: $\Pi' \leq_P \Pi$
5. A problem Π is NP-complete if $\Pi \in NP$ and Π is NP-hard

Theorem

3-SAT, CLIQUE, VERTEX-COVER... are NP-Complete.

Can sometimes show that a problem is hard to approximate within a certain factor. For example, in the homework question about locating stores in various towns you essentially showed that beating a factor 2 approximation for the problem would solve DOMINATING-SET.

Approx Algorithms and Reductions: Cautionary Tale!

Suppose $\Pi' \leq_P \Pi$ and we have an polynomial time α-approximation for a Π, do we necessarily have an α approximation for Π?

Approx Algorithms and Reductions: Cautionary Tale!

Suppose $\Pi' \leq_P \Pi$ and we have an polynomial time α-approximation for a Π, do we necessarily have an α approximation for Π?

Problem: INDEPENDENT-SET

- Input: An undirected graph $G = (V, E)$.
- Output: A set $U \subset V$ of maximum size such that no two vertices in U are connected by a single edge.

Approx Algorithms and Reductions: Cautionary Tale!

Suppose $\Pi' \leq_P \Pi$ and we have an polynomial time α-approximation for a Π, do we necessarily have an α approximation for Π?

Problem: INDEPENDENT-SET

- Input: An undirected graph $G = (V, E)$.
- Output: A set $U \subset V$ of maximum size such that no two vertices in U are connected by a single edge.

Lemma

INDEPENDENT-SET $\leq_P$ *VERTEX-COVER*

Approx Algorithms and Reductions: Cautionary Tale!

Suppose $\Pi' \leq_P \Pi$ and we have an polynomial time α-approximation for a Π, do we necessarily have an α approximation for Π?

Problem: INDEPENDENT-SET

- Input: An undirected graph $G = (V, E)$.
- Output: A set $U \subset V$ of maximum size such that no two vertices in U are connected by a single edge.

Lemma

INDEPENDENT-SET $\leq_P$ VERTEX-COVER

Proof.

$U \subset V$ is an independent set iff $V - U$ is a vertex cover. So an instance of (G, k) of INDEPENDENT-SET is a "yes" instance iff the instance $(G, n - k)$ of VERTEX-COVER is a "yes" instance. □

Approx Algorithms and Reductions: Cautionary Tale!

Suppose $\Pi' \leq_P \Pi$ and we have an polynomial time α-approximation for a Π, do we necessarily have an α approximation for Π?

Problem: INDEPENDENT-SET

- Input: An undirected graph $G = (V, E)$.
- Output: A set $U \subset V$ of maximum size such that no two vertices in U are connected by a single edge.

Lemma

INDEPENDENT-SET $\leq_P$ VERTEX-COVER

Proof.

$U \subset V$ is an independent set iff $V - U$ is a vertex cover. So an instance of (G, k) of INDEPENDENT-SET is a "yes" instance iff the instance $(G, n-k)$ of VERTEX-COVER is a "yes" instance. □

But using a factor 2-approx for Vertex-Cover may give a factor $\Omega(n)$ approximation for Independent-Set. E.g., in a perfect matching, picking $U = V$ is a 2-approx to min vertex cover and $V - U$ is an independent set of size 0. However, there's an independent set of size $|V|/2$.

And finally...

Good luck with the exam!