Part I

Knapsack
Knapsack Problem

**Input**  Given a Knapsack of capacity $W$ lbs. and $n$ objects with $i$th object having weight $w_i$ and value $v_i$; assume $W$, $w_i$, $v_i$ are all positive integers

**Goal**  Fill the Knapsack without exceeding weight limit while maximizing value.
Knapsack Problem

**Input**  Given a Knapsack of capacity $W$ lbs. and $n$ objects with $i$th object having weight $w_i$ and value $v_i$; assume $W$, $w_i$, $v_i$ are all positive integers

**Goal**  Fill the Knapsack without exceeding weight limit while maximizing value.

We saw that:

- Knapsack can be solved exactly in $O(nW)$ time via dynamic programming. Not polynomial time when $W$ is large compared to $n$.
- Knapsack is NP-Complete
If $W = 11$, the best is $\{3, 4\}$ giving value 40.
Greedy Approximation Algorithm

- Sort objects in decreasing order of $v_i/w_i$ (bang per buck)
Greedy Approximation Algorithm

- Sort objects in decreasing order of $v_i/w_i$ (bang per buck)
- Insert items in sorted order and item to knapsack if sufficient weight left.

**Bad example**: Two items:
$v_1 = 1, w_1 = 1,$ $v_2 = W - 1, w_2 = W$. Greedy will pack item 1 and stop and get value 1 while optimum solution is to pack item 2 of value $W - 1$.

**Is Greedy really bad?**

**Lemma**
If all items have weight less than $\epsilon W$ for some $\epsilon < 1$ then Greedy outputs a solution of value at least $(1 - \epsilon) \cdot \text{OPT}$. 
Greedy Approximation Algorithm

- Sort objects in decreasing order of $v_i/w_i$ (bang per buck)
- Insert items in sorted order and item to knapsack if sufficient weight left.
- Bad example: Two items: $v_1 = 1$, $w_1 = 1$, $v_2 = W - 1$, $w_2 = W$. Greedy will pack item 1 and stop and get value 1 while optimum solution is to pack item 2 of value $W - 1$. 

Is Greedy really bad?

Lemma

If all items have weight less than $\epsilon W$ for some $\epsilon < 1$ then Greedy outputs a solution of value at least $(1 - \epsilon) \text{OPT}$.
Greedy Approximation Algorithm

- Sort objects in decreasing order of $v_i/w_i$ (bang per buck)
- Insert items in sorted order and item to knapsack if sufficient weight left.
- Bad example: Two items: $v_1 = 1, w_1 = 1, v_2 = W - 1, w_2 = W$. Greedy will pack item 1 and stop and get value 1 while optimum solution is to pack item 2 of value $W - 1$. 

Lemma
If all items have weight less than $\epsilon W$ for some $\epsilon < 1$ then Greedy outputs a solution of value at least $(1 - \epsilon)OPT.$
Greedy Approximation Algorithm

- Sort objects in decreasing order of $v_i/w_i$ (bang per buck)
- Insert items in sorted order and item to knapsack if sufficient weight left.
- Bad example: Two items: $v_1 = 1, w_1 = 1, v_2 = W - 1, w_2 = W$. Greedy will pack item 1 and stop and get value 1 while optimum solution is to pack item 2 of value $W - 1$.

Is Greedy really bad?
Greedy Approximation Algorithm

- Sort objects in decreasing order of $v_i/w_i$ (bang per buck)
- Insert items in sorted order and item to knapsack if sufficient weight left.
- Bad example: Two items: $v_1 = 1$, $w_1 = 1$, $v_2 = W - 1$, $w_2 = W$. Greedy will pack item 1 and stop and get value 1 while optimum solution is to pack item 2 of value $W - 1$.

Is Greedy really bad?

**Lemma**

*If all items have weight less than $\epsilon W$ for some $\epsilon < 1$ then Greedy outputs a solution of value at least $(1 - \epsilon)OPT$.***
Pick the better of the two solutions below:

- The solution of Greedy
- The heaviest value item
Modified Greedy

Pick the better of the two solutions below:

- The solution of Greedy
- The heaviest value item

Lemma

*Modified Greedy outputs a solution of value at least $\text{OPT}/2$.***
Knapsack

Modified Greedy

Pick the better of the two solutions below:

- The solution of Greedy
- The heaviest value item

**Lemma**

*Modified Greedy outputs a solution of value at least \( \text{OPT}/2 \).*

Can we do better?
Partial Enumeration and Greedy

Let $k$ be some fixed integer.

```plaintext
current-best = 0
for each subset $S$ of $k$ items do
    if $S$ is not feasible in knapsack
        continue
    include $S$ in knapsack
    let $W' = W - \sum_{i \in S} w_i$ (* remaining capacity *)
    run Greedy on remaining items in knapsack of capacity $W'$
    if value of solution is better than current-best
        current-best = value of new solution
end for
```
Knapsack

Partial Enumeration and Greedy

Let $k$ be some fixed integer.

current-best = 0
for each subset $S$ of $k$ items do
    if $S$ is not feasible in knapsack
        continue
    include $S$ in knapsack
    let $W' = W - \sum_{i \in S} w_i$ (* remaining capacity *)
    run Greedy on remaining items in knapsack of capacity $W'$
    if value of solution is better than current-best
        current-best = value of new solution
end for

Lemma

Algorithm can be implemented in $O(n^{k+1})$ time. The algorithm outputs a solution of value at least $OPT(1 - 1/k)$. 
Theorem

For the Knapsack problem, for any fixed $\epsilon > 0$, there is a $n^{O(1/\epsilon)}$-time algorithm that has an approximation ratio of $(1 - \epsilon)$. 

Knapsack has a polynomial time approximation scheme (PTAS). It is a scheme because the algorithm for each $\epsilon > 0$ is (slightly) different. Running time of algorithm for $\epsilon = 1/10$ is $O(n^{11})$; not great. Can we do better?
Theorem

For the Knapsack problem, for any fixed $\epsilon > 0$, there is a $n^{O(1/\epsilon)}$-time algorithm that has an approximation ratio of $(1 - \epsilon)$.

Knapsack has a polynomial time approximation scheme (PTAS). It is a scheme because the algorithm for each $\epsilon > 0$ is (slightly) different.
Theorem

For the Knapsack problem, for any fixed $\epsilon > 0$, there is a $n^{O(1/\epsilon)}$-time algorithm that has an approximation ratio of $(1 - \epsilon)$.

Knapsack has a polynomial time approximation scheme (PTAS). It is a scheme because the algorithm for each $\epsilon > 0$ is (slightly) different.

Running time of algorithm for $\epsilon = 1/10$ is $O(n^{11})$; not great.
For the Knapsack problem, for any fixed $\epsilon > 0$, there is a $n^{O(1/\epsilon)}$-time algorithm that has an approximation ratio of $(1 - \epsilon)$.

Knapsack has a polynomial time approximation scheme (PTAS). It is a scheme because the algorithm for each $\epsilon > 0$ is (slightly) different.

Running time of algorithm for $\epsilon = 1/10$ is $O(n^{11})$; not great.

Can we do better?
Theorem

For the Knapsack problem, for any fixed $\epsilon > 0$, there is an algorithm that runs in $O(n \log \frac{1}{\epsilon} + \frac{1}{\epsilon^4})$ and has an approximation ratio of $(1 - \epsilon)$.

The running time of algorithm is polynomial in both $n$ and $1/\epsilon$. Such an approximation scheme is called a fully-polynomial time approximate scheme (FPTAS). This is the best we can hope for in terms of an NP-Hard optimization problem if $P \neq NP$. 
For the Knapsack problem, for any fixed $\epsilon > 0$, there is an algorithm that runs in $O(n \log \frac{1}{\epsilon} + \frac{1}{\epsilon^4})$ and has an approximation ratio of $(1 - \epsilon)$.

The running time of algorithm is polynomial in both $n$ and $1/\epsilon$. Such an approximation scheme is called a fully-polynomial time approximate scheme (FPTAS). This is the best we can hope for in terms of an NP-Hard optimization problem if $P \neq NP$.

Knapsack is “easy” in theory and practice even though it is NP-Complete.
Part II

Set Cover
(Weighted) Set Cover Problem

**Input** Given a set $U$ of $n$ elements, a collection $S_1, S_2, \ldots S_m$ of subsets of $U$, with weights $w_i$

**Goal** Find a collection $C$ of these sets $S_i$ whose union is equal to $U$ and such that $\sum_{i \in C} w_i$ is minimized.
(Weighted) Set Cover Problem

Input  Given a set $U$ of $n$ elements, a collection $S_1, S_2, \ldots S_m$ of subsets of $U$, with weights $w_i$

Goal  Find a collection $C$ of these sets $S_i$ whose union is equal to $U$ and such that $\sum_{i \in C} w_i$ is minimized.

Example
Let $U = \{1, 2, 3, 4, 5, 6, 7, 8\}$, with

$S_1 = \{1\} \quad w_1 = 1 \quad S_2 = \{2\} \quad w_2 = 1$
$S_3 = \{3, 4\} \quad w_3 = 1 \quad S_4 = \{5, 6, 7, 8\} \quad w_4 = 1$
$S_5 = \{1, 3, 5, 7\} \quad w_5 = 1 + \epsilon \quad S_6 = \{2, 4, 6, 8\} \quad w_6 = 1 + \epsilon$
(Weighted) Set Cover Problem

**Input**  Given a set $U$ of $n$ elements, a collection $S_1, S_2, \ldots S_m$ of subsets of $U$, with weights $w_i$

**Goal**  Find a collection $C$ of these sets $S_i$ whose union is equal to $U$ and such that $\sum_{i \in C} w_i$ is minimized.

**Example**
Let $U = \{1, 2, 3, 4, 5, 6, 7, 8\}$, with

- $S_1 = \{1\}, w_1 = 1$
- $S_2 = \{2\}, w_2 = 1$
- $S_3 = \{3, 4\}, w_3 = 1$
- $S_4 = \{5, 6, 7, 8\}, w_4 = 1$
- $S_5 = \{1, 3, 5, 7\}, w_5 = 1 + \epsilon$
- $S_6 = \{2, 4, 6, 8\}, w_6 = 1 + \epsilon$

$\{S_5, S_6\}$ is a set cover of weight $2 + 2\epsilon$
Greedy Rule

- Pick the next set in the cover to be the one that makes “most progress” towards the goal
Greedy Rule

- Pick the next set in the cover to be the one that makes “most progress” towards the goal
  - Covers many (uncovered) elements
Greedy Rule

- Pick the next set in the cover to be the one that makes “most progress” towards the goal
  - Covers many (uncovered) elements
  - Has a small weight
Greedy Rule

- Pick the next set in the cover to be the one that makes “most progress” towards the goal
  - Covers many (uncovered) elements
  - Has a small weight
- If $R$ is the set of elements that aren’t covered as yet, add set $S_i$ to the cover, if it minimizes the quantity $\frac{w_i}{|S_i \cap R|}$; that is the set that maximizes the ratio of weight to number of new elements covered.
Greedy Rule

- Pick the next set in the cover to be the one that makes “most progress” towards the goal
  - Covers many (uncovered) elements
  - Has a small weight
- If $R$ is the set of elements that aren’t covered as yet, add set $S_i$ to the cover, if it minimizes the quantity $\frac{w_i}{|S_i \cap R|}$; that is the set that maximizes the ratio of weight to number of new elements covered.
- If all $w_i = 1$ then greedy picks the next set that covers the max number of uncovered elements.
Greedy Algorithm

Initially \( R = U \) and \( C = \emptyset \)

while \( R \neq \emptyset \)

\[
\begin{align*}
\text{let } S_i & \text{ be the set that minimizes } w_i / |S_i \cap R| \\
C &= C \cup \{i\} \\
R &= R \setminus S_i
\end{align*}
\]

return \( C \)

Running Time

Main loop iterates for \( O(n) \) time, where \(|U| = n\)

Minimum \( S_i \) can be found in \( O(\log m) \) time, using a priority heap, where there are \( m \) sets in set cover instance

Total time is \( O(n \log m) \)
Greedy Algorithm

Initially $R = U$ and $C = \emptyset$

while $R \neq \emptyset$

    let $S_i$ be the set that minimizes $w_i/|S_i \cap R|$

    $C = C \cup \{i\}$

    $R = R \setminus S_i$

return $C$

Running Time

Main loop iterates for $O(n)$ time, where $|U| = n$

Minimum $S_i$ can be found in $O(\log m)$ time, using a priority heap, where there are $m$ sets in set cover instance

Total time is $O(n \log m)$
Greedy Algorithm

Initially $R = U$ and $C = \emptyset$

while $R \neq \emptyset$

    let $S_i$ be the set that minimizes $w_i/|S_i \cap R|$
    $C = C \cup \{i\}$
    $R = R \setminus S_i$

return $C$

Running Time

- Main loop iterates for $O(n)$ time, where $|U| = n$
Greedy Algorithm

Initially \( R = U \) and \( C = \emptyset \)
while \( R \neq \emptyset \)
    let \( S_i \) be the set that minimizes \( w_i / |S_i \cap R| \)
    \( C = C \cup \{i\} \)
    \( R = R \setminus S_i \)
return \( C \)

Running Time

- Main loop iterates for \( O(n) \) time, where \( |U| = n \)
Greedy Algorithm

Initially $R = U$ and $C = \emptyset$
while $R \neq \emptyset$
    let $S_i$ be the set that minimizes $w_i / |S_i \cap R|$
    $C = C \cup \{i\}$
    $R = R \setminus S_i$
return $C$

Running Time
- Main loop iterates for $O(n)$ time, where $|U| = n$
- Minimum $S_i$ can be found in $O(\log m)$ time, using a priority heap, where there are $m$ sets in set cover instance
Greedy Algorithm

Initially \( R = U \) and \( C = \emptyset \)
while \( R \neq \emptyset \)
    let \( S_i \) be the set that minimizes \( w_i / |S_i \cap R| \)
    \( C = C \cup \{i\} \)
    \( R = R \setminus S_i \)
return \( C \)

Running Time

- Main loop iterates for \( O(n) \) time, where \( |U| = n \)
- Minimum \( S_i \) can be found in \( O(\log m) \) time, using a priority heap, where there are \( m \) sets in set cover instance
- Total time is \( O(n \log m) \)
Example: Greedy Algorithm

Let $U = \{1, 2, 3, 4, 5, 6, 7, 8\}$, with

$S_1 = \{1\}$, $S_2 = \{2\}$,
$S_3 = \{3, 4\}$, $S_4 = \{5, 6, 7, 8\}$,
$S_5 = \{1, 3, 5, 7\}$, $S_6 = \{2, 4, 6, 8\}$

$w_1 = w_2 = w_3 = w_4 = 1$ and $w_5 = w_6 = 1 + \epsilon$

Greedy Algorithm first picks $S_4$, then $S_3$, and finally $S_1$ and $S_2$
Analysis of the Greedy Algorithm

\[ H(k): \text{k’th harmonic number. } H(k) = 1 + 1/2 + \ldots + 1/k \approx \ln k. \]

**Theorem**

*The greedy algorithm for set cover is a \( H(d^*) \)-approximation, where \( d^* = \max_i |S_i| \).*
Analysis of the Greedy Algorithm

$H(k)$: k’th harmonic number. $H(k) = 1 + 1/2 + \ldots + 1/k \approx \ln k$.

**Theorem**

*The greedy algorithm for set cover is a $H(d^*)$-approximation, where $d^* = \max_i |S_i|$*

**Analysis Tight?**

Does the Greedy Algorithm give better approximation guarantees?
Set Cover

The Problem
Greedy Heuristic
Analysis of Greedy Algorithm

Analysis of the Greedy Algorithm

\[ H(k): \text{k'\text{th harmonic number. } } H(k) = 1 + \frac{1}{2} + \ldots + \frac{1}{k} \approx \ln k. \]

**Theorem**

*The greedy algorithm for set cover is a \( H(d^*) \)-approximation, where \( d^* = \max_i |S_i| \)*

**Analysis Tight?**

Does the Greedy Algorithm give better approximation guarantees? No!

Consider a generalization of the set cover example. Each column has \( 2^{k-1} \) elements, and there are two sets consisting of a column each with weight \( 1 + \epsilon \). Additionally there are \( \log n \) sets of increasing size of weight 1. The greedy algorithm will pick these \( \log n \) sets given weight \( \log \log n \), while the best cover has weight \( 2 + 2\epsilon \).
Best Algorithm for Set Cover

**Theorem**

*If* $P \neq NP$ *then no polynomial time algorithm can achieve a better than* $H(n)$ *approximation.*

Proof beyond the scope of this course.
Part III

Vertex Cover
(Weighted) Vertex Cover

**Input**  Given graph $G = (V, E)$ with weights $w_i \geq 0$ associated with each vertex $i$

**Goal**  Find a vertex cover $S \subseteq V$ such that $\sum_{i \in S} w_i$ is minimized

![Graph Illustration]

{D} has weight 9, while {A, B, C} has weight 8
(Weighted) Vertex Cover

**Input**  Given graph $G = (V, E)$ with weights $w_i \geq 0$ associated with each vertex $i$

**Goal**  Find a vertex cover $S \subseteq V$ such that $\sum_{i \in S} w_i$ is minimized

**Figure:** Vertex Cover $\{D\}$ has weight 9, while $\{A, B, C\}$ has weight 8
Unweighted Case

Question
When all sets have weight 1 and all vertices have weight 1, we know Vertex Cover \( \leq_P \) Set Cover. Can we use the approximation algorithm for Set Cover to get an approximation algorithm for Vertex Cover?
Unweighted Case

**Question**

When all sets have weight 1 and all vertices have weight 1, we know $\text{Vertex Cover} \leq_P \text{Set Cover}$

Can we use the approximation algorithm for Set Cover to get an approximation algorithm for Vertex Cover? Yes, but not true for all reductions
Approximating Vertex Cover using Set Cover

**Theorem**

*There is an $H(d)$-approximation algorithm for Vertex Cover, where $d$ is the maximum degree of any vertex.*
Approximating Vertex Cover using Set Cover

**Theorem**

There is an $H(d)$-approximation algorithm for Vertex Cover, where $d$ is the maximum degree of any vertex.

**Proof.**

The approximation algorithm uses the reduction to set cover.
Approximating Vertex Cover using Set Cover

**Theorem**

There is an $H(d)$-approximation algorithm for Vertex Cover, where $d$ is the maximum degree of any vertex.

**Proof.**

The approximation algorithm uses the reduction to set cover.

- The universe (for set cover problem) is the set of edges.
Approximating Vertex Cover using Set Cover

Theorem

There is an $H(d)$-approximation algorithm for Vertex Cover, where $d$ is the maximum degree of any vertex

Proof.

The approximation algorithm uses the reduction to set cover

- The universe (for set cover problem) is the set of edges
- For each vertex $i$, create $S_i$ of edges incident upon $i$, with weight $w_i$
Approximating Vertex Cover using Set Cover

**Theorem**

There is an $H(d)$-approximation algorithm for Vertex Cover, where $d$ is the maximum degree of any vertex

**Proof.**

The approximation algorithm uses the reduction to set cover

- The universe (for set cover problem) is the set of edges
- For each vertex $i$, create $S_i$ of edges incident upon $i$, with weight $w_i$
- $C$ is a set cover of weight $w$ iff $C$ is a vertex cover of weight $w$
Approximating Vertex Cover using Set Cover

Theorem

There is an $H(d)$-approximation algorithm for Vertex Cover, where $d$ is the maximum degree of any vertex.

Proof.

The approximation algorithm uses the reduction to set cover:

- The universe (for set cover problem) is the set of edges.
- For each vertex $i$, create $S_i$ of edges incident upon $i$, with weight $w_i$.
- $C$ is a set cover of weight $w$ iff $C$ is a vertex cover of weight $w$.
- The desired approximation algorithm runs the algorithm for set cover on the reduced instance.
Greedy for Vertex Cover

Reduction essentially says to run following greedy algorithm:

- Initialize $S$ to $\emptyset$.
- While graph has edges left
  - Pick vertex $v$ with highest degree and add it to $S$
  - Remove $v$ and all edges incident to $v$ from graph
- Output $S$
Polynomial reduction does not imply approximation
Polynomial reduction does not imply approximation

- Independent Set $\leq_P$ Vertex Cover
Polynomial reduction does not imply approximation

- Independent Set $\leq_P$ Vertex Cover
  - $V \setminus I$ is a vertex cover iff $I$ is independent set
Polynomial reduction does not imply approximation

- Independent Set $\leq_P$ Vertex Cover
  - $V \setminus I$ is a vertex cover iff $I$ is independent set
- Suppose we have a 2-approximation alg $A$ for vertex cover
Reducions and Approximation Algorithms

Polynomial reduction does not imply approximation

- Independent Set $\leq_P$ Vertex Cover
  - $V \setminus I$ is a vertex cover iff $I$ is independent set
- Suppose we have a 2-approximation alg $A$ for vertex cover
- Suppose $G = (V, E)$ has vertex cover of size $|V|/2$
Polynomial reduction does not imply approximation

- Independent Set \( \leq_P \) Vertex Cover
  - \( V \setminus I \) is a vertex cover iff \( I \) is independent set
- Suppose we have a 2-approximation alg \( A \) for vertex cover
- Suppose \( G = (V, E) \) has vertex cover of size \( |V|/2 \)
- So \( A \) could return \( V \) as vertex cover on \( G \)
Reductions and Approximation Algorithms

Polynomial reduction does not imply approximation

- Independent Set $\leq_P$ Vertex Cover
  - $V \setminus I$ is a vertex cover iff $I$ is independent set
- Suppose we have a 2-approximation alg $A$ for vertex cover
- Suppose $G = (V, E)$ has vertex cover of size $|V|/2$
- So $A$ could return $V$ as vertex cover on $G$
- $V \setminus V = \emptyset$ is not a good approximation of an independent set of size $|V|/2$
Back to Vertex Cover

Greedy algorithm given an $H(n) \simeq \ln n$ approximation.
Greedy algorithm given an $H(n) \approx \ln n$ approximation.

- Does Greedy actually give a better approximation?
Greedy algorithm given an $H(n) \approx \ln n$ approximation.

- Does Greedy actually give a better approximation? No!. There are examples which show that Greedy is no better than $H(n)$ approx.
Greedy algorithm given an $H(n) \approx \ln n$ approximation.

- Does Greedy actually give a better approximation? No!. There are examples which show that Greedy is no better than $H(n)$ approx.
- Is there a better approximation algorithm for Vertex Cover?
Back to Vertex Cover

Greedy algorithm given an $H(n) \simeq \ln n$ approximation.

- Does Greedy actually give a better approximation? No!. There are examples which show that Greedy is no better than $H(n)$ approx.
- Is there a better approximation algorithm for Vertex Cover? Yes! We will see a 2-approximation.
Vertex Cover as Linear Constraints

For a graph $G = (V, E)$ with vertex weights $w_i$, we have variables $x_i$, which will be either 0 or 1, indicating whether vertex $i$ is part of the cover.

Minimize $\sum_{i \in V} w_i x_i$

subject to $x_i + x_j \geq 1$ for each $(i, j) \in E$

$x_i \in \{0, 1\}$ for each $i \in V$
0-1 Integer Linear Programming

Given an objective function, and a collection of linear constraints, find an assignment of 0-1 values to the variables such that all linear constraints are satisfied and the objective function is optimized.
0-1 Integer Linear Programming

**Integer Programming**

Given an objective function, and a collection of linear constraints, find an assignment of 0-1 values to the variables such that all linear constraints are satisfied and the objective function is optimized.

**Theorem**

0-1 Integer linear programming is NP-complete
LP Relaxation

The linear programming relaxation of an Integer Linear Program is obtained by removing the constraint that the variables be integers.

Minimize \( \sum_{i \in V} w_i x_i \)
subject to \( x_i + x_j \geq 1 \) for each \((i, j) \in E\)
\( x_i \geq 0 \) for each \( i \in V \)
The linear programming relaxation of an Integer Linear Program is obtained by removing the constraint that the variables be integers

Minimize \[ \sum_{i \in V} w_i x_i \]
subject to \[ x_i + x_j \geq 1 \text{ for each } (i, j) \in E \]
\[ x_i \geq 0 \text{ for each } i \in V \]

**Proposition**

If \( x^1 \) is the optimal solution for the LP relaxation, and \( x^2 \) is the optimal solution to the ILP, then \[ \sum_{i \in V} w_i x_i^1 \leq \sum_{i \in V} w_i x_i^2 \]
LP Relaxation: Example

All vertices have weight 1.
LP Relaxation: Example

All vertices have weight 1. Any vertex cover must have at least 2 vertices, and hence weight 2.
LP Relaxation: Example

All vertices have weight 1. Any vertex cover must have at least 2 vertices, and hence weight 2.
LP Relaxation: Example

Min $x_1 + x_2 + x_3$

s.t. $x_1 + x_2 \geq 1$
    $x_2 + x_3 \geq 1$
    $x_3 + x_1 \geq 1$
    $x_1, x_2, x_3 \geq 0$

All vertices have weight 1. Any vertex cover must have at least 2 vertices, and hence weight 2.
The LP problem has solution $x_1 = x_2 = x_3 = 1/2$ whose value is $3/2$. 
Vertex Cover: LP Relaxation

Weighted vertex cover is the ILP

Min \sum_{i \in V} w_i x_i \\
\text{s.t. } x_i + x_j \geq 1 \quad (i, j) \in E \\
x_i \in \{0, 1\} \quad i \in V
Vertex Cover: LP Relaxation

Weighted vertex cover is the ILP

\[
\begin{align*}
\text{Min} & \quad \sum_{i \in V} w_i x_i \\
\text{s.t.} & \quad x_i + x_j \geq 1 \quad (i, j) \in E \\
& \quad x_i \in \{0, 1\} \quad i \in V
\end{align*}
\]

Its LP-relaxation is

\[
\begin{align*}
\text{Min} & \quad \sum_{i \in V} w_i x_i \\
\text{s.t.} & \quad x_i + x_j \geq 1 \quad (i, j) \in E \\
& \quad x_i \geq 0 \quad i \in V
\end{align*}
\]
Vertex Cover: LP Relaxation

Weighted vertex cover is the ILP

\[
\begin{align*}
\text{Min} & \quad \sum_{i \in V} w_i x_i \\
\text{s.t.} & \quad x_i + x_j \geq 1 \quad (i, j) \in E \\
& \quad x_i \in \{0, 1\} \quad i \in V
\end{align*}
\]

Its LP-relaxation is

\[
\begin{align*}
\text{Min} & \quad \sum_{i \in V} w_i x_i \\
\text{s.t.} & \quad x_i + x_j \geq 1 \quad (i, j) \in E \\
& \quad x_i \geq 0 \quad i \in V
\end{align*}
\]

• Solutions to the LP don’t correspond to vertex covers, because variables may have fractional values.
Weighted vertex cover is the ILP

$$\text{Min} \quad \sum_{i \in V} w_i x_i$$

s.t. \quad \begin{align*}
  x_i + x_j & \geq 1 \quad (i, j) \in E \\
  x_i & \in \{0, 1\} \quad i \in V
\end{align*}

Its LP-relaxation is

$$\text{Min} \quad \sum_{i \in V} w_i x_i$$

s.t. \quad \begin{align*}
  x_i + x_j & \geq 1 \quad (i, j) \in E \\
  x_i & \geq 0 \quad i \in V
\end{align*}

- Solutions to the LP don’t correspond to vertex covers, because variables may have fractional values.
- Can solving the LP-relaxation, nonetheless, help?
LP rounding

Algorithm

Solve LP relaxation optimally to get solution $x^*$

Round the fraction values to obtain a solution to the vertex cover problem, i.e., $S = \{i | x^*_i \geq 1/2\}$

Challenges

Is $S$ obtained by rounding guaranteed to be a vertex cover?

How large is $w(S)$ compared to optimal cover?
LP rounding

Algorithm

1. Solve LP relaxation optimally to get solution $x^*$
LP rounding

Algorithm

1. Solve LP relaxation optimally to get solution $x^*$

2. Round the fraction values to obtain a solution to the vertex cover problem, i.e., $S = \{ i \mid x^*_i \geq 1/2 \}$
Algorithm

1. Solve LP relaxation optimally to get solution $x^*$
2. Round the fraction values to obtain a solution to the vertex cover problem, i.e., $S = \{i \mid x_i^* \geq 1/2\}$
3. return $S$
LP rounding

**Algorithm**

1. Solve LP relaxation optimally to get solution $x^*$
2. Round the fraction values to obtain a solution to the vertex cover problem, i.e., $S = \{ i \mid x_i^* \geq 1/2 \}$
3. return $S$

**Challenges**

- Is $S$ obtained by rounding, guaranteed to be a vertex cover?
- How large is $w(S)$ compared to optimal cover?
LP rounding

Algorithm

1. Solve LP relaxation optimally to get solution $x^*$
2. Round the fraction values to obtain a solution to the vertex cover problem, i.e., $S = \{i \mid x_i^* \geq 1/2\}$
3. return $S$

Challenges

- Is $S$ obtained by rounding, guaranteed to be a vertex cover?
LP rounding

Algorithm
1. Solve LP relaxation optimally to get solution $x^*$
2. Round the fraction values to obtain a solution to the vertex cover problem, i.e., $S = \{ i \mid x_i^* \geq 1/2 \}$
3. return $S$

Challenges
- Is $S$ obtained by rounding, guaranteed to be a vertex cover?
- How large is $w(S)$ compared to optimal cover?
Correctness of LP rounding

Lemma

Set $S$ obtained by rounding the LP-relaxation is a vertex cover.
Correctness of LP rounding

Lemma

Set $S$ obtained by rounding the LP-relaxation is a vertex cover

Proof.

Consider any edge $e = (i, j)$. Since $x_i^* + x_j^* \geq 1$, we know $x_i^* \geq 1/2$ or $x_j^* \geq 1/2$. Thus, either $i$ or $j$ is in $S$. Therefore, $S$ is a vertex cover.
Correctness of LP rounding

Lemma

Set $S$ obtained by rounding the LP-relaxation is a vertex cover

Proof.

- Consider any edge $e = (i, j)$
Correctness of LP rounding

Lemma

*Set $S$ obtained by rounding the LP-relaxation is a vertex cover*

Proof.

- Consider any edge $e = (i,j)$
- Since $x_i^* + x_j^* \geq 1$, we know $x_i^* \geq 1/2$ or $x_j^* \geq 1/2$
Correctness of LP rounding

Lemma

Set $S$ obtained by rounding the LP-relaxation is a vertex cover.

Proof.

- Consider any edge $e = (i, j)$
- Since $x_i^* + x_j^* \geq 1$, we know $x_i^* \geq 1/2$ or $x_j^* \geq 1/2$
- Thus, either $i$ or $j$ is in $S$
Lemma

Let $w^*_{LP}$ be the optimal value of the LP relaxation and let $w^*$ be the weight of an optimum vertex cover. Then $w^*_{LP} \leq w^*$. 
**Lower Bound provided by LP Relaxation**

**Lemma**

Let $w_{LP}^*$ be the optimal value of the LP relaxation and let $w^*$ be the weight of an optimum vertex cover. Then $w_{LP}^* \leq w^*$.

**Proof.**

Let $S^*$ be an optimum vertex cover with weight $w^*$. Consider a feasible solution $x$ to LP where $x_i = 1$ if $i \in S$ and $x_i = 0$ otherwise. $wx = w(S^*) = w^*$. Therefore optimum value of LP can be no more than $w^*$. 

Chekuri CS473ug
Lower Bound provided by LP Relaxation

Lemma

Let $w_{LP}^*$ be the optimal value of the LP relaxation and let $w^*$ be the weight of an optimum vertex cover. Then $w_{LP}^* \leq w^*$.

Proof.

Let $S^*$ be an optimum vertex cover with weight $w^*$. Consider a feasible solution $x$ to LP where $x_i = 1$ if $i \in S$ and $x_i = 0$ otherwise. $wx = w(S^*) = w^*$. Therefore optimum value of LP can be no more than $w^*$.

Thus solving the LP gives us a lower bound on the weight on $w^*$. This is often useful in also back-tracking heuristics that solve the problem exactly in exponential time.
Approximation Guarantee

Lemma

Weight of vertex cover $S$ return by LP rounding is at most 2-times the weight of the optimal cover
## Approximation Guarantee

**Lemma**

*Weight of vertex cover \( S \) return by LP rounding is at most 2-times the weight of the optimal cover.*

**Proof.**

Let \( w_{LP}^* \) be the optimal value for the LP, and let \( S \) be cover obtained by rounding \( x^* \). Let \( S^* \) be the optimum vertex cover.

Let \( w^* \) be the weight of \( S^* \). Then, we have:

\[
0 \leq \frac{w^*}{2} \leq \frac{w_{LP}^*}{2}
\]

Since \( x^* \geq \frac{1}{2} \) for every \( i \in S \), we have:

\[
\frac{w^*}{2} \leq \sum_{i \in S} w_i x^*_i \leq \frac{w_{LP}^*}{2} \leq 2w^*.
\]
Approximation Guarantee

**Lemma**

Weight of vertex cover $S$ return by LP rounding is at most 2-times the weight of the optimal cover

**Proof.**

Let $w^*_{LP}$ be the optimal value for the LP, and let $S$ be cover obtained by rounding $x^*$. Let $S^*$ be the optimum vertex cover

- $w^*_{LP} \leq w(S^*)$ from lemma.
Approximation Guarantee

**Lemma**

Weight of vertex cover $S$ return by LP rounding is at most 2-times the weight of the optimal cover.

**Proof.**

Let $w^*_{LP}$ be the optimal value for the LP, and let $S$ be cover obtained by rounding $x^*$. Let $S^*$ be the optimum vertex cover

- $w^*_{LP} \leq w(S^*)$ from lemma.
- $w^*_{LP} = \sum_i w_i x^*_i \geq \sum_{i \in S} w_i x^*_i$
**Lemma**

Weight of vertex cover $S$ return by LP rounding is at most 2-times the weight of the optimal cover

**Proof.**

Let $w^*_{LP}$ be the optimal value for the LP, and let $S$ be cover obtained by rounding $x^*$. Let $S^*$ be the optimum vertex cover

- $w^*_{LP} \leq w(S^*)$ from lemma.
- $w^*_{LP} = \sum_i w_i x^*_i \geq \sum_{i \in S} w_i x^*_i$
- Since $x^*_i \geq 1/2$ for every $i \in S$, we have
  $w^*_{LP} \geq \sum_{i \in S} w_i x^*_i \geq (1/2) \sum_{i \in S} w_i = (1/2)w(S)$
Approximation Guarantee

Lemma

Weight of vertex cover $S$ return by LP rounding is at most 2-times the weight of the optimal cover

Proof.

Let $w_{LP}^*$ be the optimal value for the LP, and let $S$ be cover obtained by rounding $x^*$. Let $S^*$ be the optimum vertex cover

- $w_{LP}^* \leq w(S^*)$ from lemma.
- $w_{LP}^* = \sum_i w_i x_i^* \geq \sum_{i \in S} w_i x_i^*$
- Since $x_i^* \geq 1/2$ for every $i \in S$, we have $w_{LP}^* \geq \sum_{i \in S} w_i x_i^* \geq (1/2) \sum_{i \in S} w_i = (1/2) w(S)$
- Hence $w(S) \leq 2w_{LP}^* \leq 2w(S^*)$. 

Chekuri CS473ug
Integer Programming and Heuristics

Using Integer Programming to solve problems is a meta-heuristic method.

- Integer programming can “naturally” model many NP-Complete problems.
- Linear programming relaxations and a variety of heuristic methods like branch-and-bound, branch-and-cut, cutting places, etc are used to solve integer programs in practice.
- Very effective for many applications.
Approximation and NP-Hard problems

- **Load balancing**: can obtain a \((1 + \epsilon)\)-approximation in \(n^{O(1/\epsilon)}\) time. PTAS
- **Knapsack**: can obtain a \((1 - \epsilon)\)-approximation in \(O(n \log 1/\epsilon + 1/\epsilon^4)\) time. FPTAS
- **Set cover**: can obtain an \(\ln n + 1\) approximation. Essentially no better approximation possible unless \(P = NP\)
- **Vertex cover**: 2-approximation. Unless \(P = NP\) cannot obtain a 1.36 approximation.
Approximation and NP-Hard problems

- Load balancing: can obtain a \((1 + \epsilon)\)-approximation in \(n^{O(1/\epsilon)}\) time. PTAS
- Knapsack: can obtain a \((1 - \epsilon)\)-approximation in \(O(n \log 1/\epsilon + 1/\epsilon^4)\) time. FPTAS
- Set cover: can obtain an \(\ln n + 1\) approximation. Essentially no better approximation possible unless \(P = NP\)
- Vertex cover: 2-approximation. Unless \(P = NP\) cannot obtain a 1.36 approximation.

**Lesson:** NP-Hard optimization problems can differ dramatically in approximation even though they are all equivalent in terms of exact solvability. Some are (much) easier than others.
Approximation Algorithms: Pros and Cons

Pros:
- Systematic and theoretically sound approach to studying heuristics for problems
- Explanation for why/how problems differ despite equivalence for exact solvability
- Allows one to explore structure of intractable problems
- Can lead to successful heuristics
- Algorithmic and mathematical elegance

Cons:
- Not applicable to decision problems such as SAT
- Worst-case approach is not ideal in some practical situations