A Randomized Approximation Algorithm (Vertex Cover)

An Approximation Algorithm (Metric TSP)

A PTAS (Subset-Sum)

Approximation Algorithms for MAX-3-CNF

A Linear Programming Based (Weighted Vertex Cover)
Background
Background

1. **Linear Algebra**
   - Matrices
   - Vectors, inner product
   - etc.

2. **Probability Theory**
   - Expectation, variance
   - Basic distributions (binomial, Poisson, exponential, etc)
   - Markov's inequality
     $\Pr[|X| \geq a] \leq \frac{E(|X|)}{a}$
   - Chebyshev's inequality
     $\Pr[|x - E(x)| \geq k] \leq \frac{\mu^2}{k^2}$
   - etc.

3. **Algorithm Techniques**
   - $O()$ notation
   - Graph algorithms, e.g., breath and depth first search, minimal spanning tree, topological sort, maximum-flow, etc.
   - P, NP, NP-completeness, NP-hard, PTAS
   - Linear programming (duality)
   - etc.
Background

1. **Linear Algebra**
   - Matrices
   - Vectors, inner product
   - etc.

2. **Probability Theory**
   - Expectation, variance
   - Basic distributions (binomial, Poisson, exponential, etc)
   - Markov’s inequality \( \Pr[|X| \geq a] \leq E(|X|)/a \)
   - Chebyshev's inequality \( \Pr[|x - E(x)| \geq k] \leq \mu^2/k^2 \)
   - etc.
Background

1. **Linear Algebra**
   - Matrices
   - Vectors, inner product
   - etc.

2. **Probability Theory**
   - Expectation, variance
   - Basic distributions (binomial, Poisson, exponential, etc)
   - Markov's inequality: \( \Pr[|X| \geq a] \leq E(|X|)/a \)
   - Chebyshev's inequality: \( \Pr[|X - E(X)| \geq k] \leq \mu^2/k^2 \)
   - etc.

3. **Algorithm Techniques**
   - \( O() \) notation
   - Graph algorithms, e.g., breath and depth first search, minimal spanning tree, topological sort, maximum-flow, etc.
   - P, NP, NP-completeness, NP-hard, PTAS
   - Linear programming (duality)
   - etc.
NP-Complete versus NP-Hard

Definition (Optimization problem)
Find a best objective, e.g., Given graph $G$, find a minimal size vertex cover.

Definition (Decision problem)
Have a yes/no answer, e.g., Given a graph $G$ and integer $k$, does $G$ have a vertex cover of size $\leq k$?

$P = ? NP$
NP-complete problems are, by definition, decision problems. If they can be solved in polynomial time then $P = NP$.

NP-complete and NP-hard
Problems that have the property that if they can be solved in polynomial time then $P = NP$, but not necessarily vice-versa, are called NP-hard. The optimization versions of NP-complete decision problems are NP-hard.
Performance Ratios for Approximation Algorithms

Let $C$ be the cost of the algorithm, let $C^*$ be the cost of an optimal solution, for any input of size $n$, the algorithm is called $\rho(n)$-approximation if $\max(C/C^*, C^*/C) \leq \rho(n)$.

**Definition (Approximation scheme)**
An **approximation scheme** for an optimization problem is an approximation algorithm that takes as input not only an instance of the problem, but also a value $\epsilon > 0$ such that for any fixed $\epsilon$, the scheme is a $(1 + \epsilon)$-approximation algorithm.

**Definition (PTAS (Polynomial-Time Approximation Scheme))**
We say an approximation scheme is a **polynomial-time approximation scheme** if for any fixed $\epsilon > 0$, the scheme runs in time polynomial in the size $n$ of its input instance.

**Definition (FPTAS (Fully Polynomial-Time Approximation Scheme))**
We say an approximation scheme is a **fully polynomial-time approximation scheme** if it is an approximation scheme and its running time is polynomial both in $1/\epsilon$ and in the size $n$ of the input instance.
A Randomized Approximation Algorithm (Vertex Cover)

An Approximation Algorithm (Metric TSP)

A PTAS (Subset-Sum)

Approximation Algorithms for MAX-3-CNF

A Linear Programming Based (Weighted Vertex Cover)
Vertex Cover

Definition (Vertex Cover problem)
A vertex cover of an undirected graph \( G = (V, E) \) is a subset of vertices \( V' \subseteq V \) such that if \( (u, v) \in E \), then either \( u \in V' \), \( v \in V' \), or both.

Algorithm 1.1: RVC\((G)\)
\[
C = \emptyset; \quad E' = E;
\]
\[
\text{while } (E' \neq \emptyset) \begin{cases} 
\text{Pick up } (u, v) \text{ from } E'; \\
C' = C \cup \{u, v\}; \\
\text{Remove every edge touching } u \text{ or } v \text{ from } E'; 
\end{cases}
\]
\[
\text{return } (C)
\]

Question
How do we analyze it?
**Definition (Vertex Cover problem)**

A *vertex cover* of an undirected graph $G = (V, E)$ is a subset of vertices $V' \subseteq V$ such that

If $(u, v) \in E$, then either $u \in V'$, $v \in V'$, or both.
Vertex Cover

Definition (Vertex Cover problem)

A vertex cover of an undirected graph $G = (V, E)$ is a subset of vertices $V' \subseteq V$ such that

If $(u, v) \in E$, then either $u \in V'$, $v \in V'$, or both.

Algorithm 1.3: RVC($G$)

$C = \emptyset$;
$E' = E$;
while ($E' \neq \emptyset$)
    \begin{align*}
    &\text{Pick up } (u, v) \text{ from } E' \text{ randomly;} \\
    &C' = C \cup \{u, v\}; \\
    &\text{Remove every edge touching } u \text{ or } v \text{ from } E'; \\
    \end{align*}
return $(C)$
Vertex Cover

Definition (Vertex Cover problem)
A vertex cover of an undirected graph \( G = (V, E) \) is a subset of vertices \( V' \subseteq V \) such that

\[
\text{If } (u, v) \in E, \text{ then either } u \in V', v \in V', \text{ or both.}
\]

Algorithm 1.4: \text{RVC}(G)

\[
\begin{align*}
C &= \emptyset; \\
E' &= E; \\
\text{while } (E' \neq \emptyset) & \quad \text{Pick up } (u, v) \text{ from } E' \text{ randomly; } \\
& \quad \{ C' = C \cup \{u, v\}; \} \\
& \quad \{ \text{Remove every edge touching } u \text{ or } v \text{ from } E'; \} \\
\text{return } (C)
\end{align*}
\]

Question
How do we analyze it?
Approximation

For any graph $G$, define $RVC(G)$ as the number of vertices chosen by algorithm $RVC$, $OPT(G)$ as the size of the smallest vertex cover of $G$.

Theorem $RVC$ runs in time $O(|V| + |E|)$.

Refer to page 1025 of CLRS.

Theorem $RVC$ is polynomial-time $2$-approximation algorithm.

$1 \leq \frac{RVC(G)}{OPT(G)} \leq 2$.

Proof. $|OPT(G)| \geq$ the number of edges pick up randomly in the loop, $= |RVC(G)|^2$. 
Approximation

For any graph $G$, define $RVC(G)$ as the number of vertices chosen by algorithm RVC, $OPT(G)$ as the size of the smallest vertex cover of $G$. 

Theorem

$RVC$ runs in time $O(|V| + |E|)$.

Refer to page 1025 of CLRS.

Theorem

$RVC$ is polynomial-time $2$-approximation algorithm.

$1 \leq \frac{RVC(G)}{OPT(G)} \leq 2$.

Proof.

$|OPT(G)| \geq$ the number of edges pick up randomly in the loop, $= |RVC(G)|$. 
Approximation

For any graph $G$, define $RVC(G)$ as the number of vertices chosen by algorithm RVC, $OPT(G)$ as the size of the smallest vertex cover of $G$.

**Theorem**

*RVC runs in time* $O(|V| + |E|)$.

Refer to page 1025 of CLRS.
Approximation

For any graph $G$, define $RVC(G)$ as the number of vertices chosen by algorithm RVC, $OPT(G)$ as the size of the smallest vertex cover of $G$.

**Theorem**

*RVC runs in time $O(|V| + |E|)$.*

Refer to page 1025 of CLRS.

**Theorem**

*RVC is polynomial-time 2-approximation algorithm.*

$1 \leq RVC(G)/OPT(G) \leq 2$.

**Proof.**

\[
|OPT(G)| \geq \text{the number of edges pick up randomly in the loop,} \\
= \frac{|RVC(G)|}{2}.
\]
Question
Is the following algorithm 2-approximation?

Algorithm 1.5: RVC(G)

\[ C = \emptyset; \]
\[ E' = E; \]
\[ \text{while } (E' \neq \emptyset) \]
\[ \begin{cases} 
\text{Select a vertex of the highest degree } v \in E'; \\
C' = C \cup v; \\
\text{Remove all } v \text{'s incident edges;}
\end{cases} \]
\[ \text{return} \ (C) \]

Hint: Try a bipartite graph with vertices of uniform degree on the left and vertices of varying degree on the right.
Consider a **special case** which is not NP-hard.

**Theorem**

*There exists an efficient algorithm (running in polynomial time) to find the optimal vertex cover if $G = (V, E)$ is a tree.*

**Proof.**

Greedy approach.

**Theorem**

*There exists an efficient algorithm (running in polynomial time) to find the optimal weighted vertex cover if $G = (V, E)$ is a tree.*

**Proof.**

Dynamic programming approach.
1. Heuristics
2. Local search
3. Simulated annealing
4. Tabu search
5. Genetic algorithms
6. etc.
A Randomized Approximation Algorithm (Vertex Cover)

An Approximation Algorithm (Metric TSP)

A PTAS (Subset-Sum)

Approximation Algorithms for MAX-3-CNF

A Linear Programming Based (Weighted Vertex Cover)
Three Graph Problems

Let $G = (V, E)$ be a graph. Each $e \in E$ has a cost $c_e$. The cost of a set of edges $E' \subseteq E$ is $c(E') = \sum_{e \in E'} c_e$. A Hamiltonian cycle is a tour (cycle) in passing through every vertex exactly once.

Definition
Hamiltonian Cycle (HC): Does $G = (V, E)$ contain a Hamiltonian cycle?

Definition
Traveling Salesman Problems (TSP): Find a Hamiltonian cycle, $T$, with minimal cost $c(T)$ among all cycles.

Definition
Euler Tour (ET): Find a path in the graph $G$ that uses each edge in the graph exactly once (a repeated vertex is used exactly as many times as it appears). An Euler tour can be found in polynomial time $O(|V| + |E|)$. 
Metric TSP

Definition
A graph $G$ with cost $c()$ has the triangle inequality if for all vertices $u, v, w \in V$

$$c(u,w) \leq c(u,v) + c(v,w).$$

Algorithm 2.1: Metric-TSP($G$)

Find a minimum spanning tree $T'$ of $G$.
Double every edge in $T'$ to get new graph $G'$.
Find an Euler tour $T'$ of $G'$.
Output the vertices of $G$ in the order in which they first appear in $T'$.
Let $T$ be the Hamiltonian cycle thus created.
return ($T$)

Lemma
Metric-TSP runs in polynomial time.
Analysis of Metric-TSP

**Theorem**

*Metric-TSP is a 2-approximation algorithm for the TSP problem on metric graphs.*

**Proof.**

**Goal:**

\[
\begin{align*}
c(T') & \leq OPT(G) \\
c(T') & = 2 \cdot C(T') \\
c(T) & \leq c(T') \\
& = 2 \cdot C(T') \\
& \leq 2 \cdot OPT(T)
\end{align*}
\]
Definition

Matching. Let $G = (V, E)$ be a graph with cost function $c(.)$ on its edges

1. A matching of $G$ is a set of edges $E' \subseteq E$ such that no two edges in $E'$ share a vertex in common

2. A perfect matching is a matching in which $|E'| = \frac{|E|}{2}$. Perfect matching of complete graphs always exist

3. A minimum-cost perfect matching of a complete graph can be found in $O(|V|^3)$ time

What is the relationship between a matching and TSP?
Algorithm 2.2: Chris-TSP\( (G) \)

Find a minimum spanning tree \( T' \) of \( G \).
Find a minimal cost perfect matching, \( M \), on the vertices of odd-degree in \( T' \).
Let \( E' \) be the union of \( T' \) and \( M \).
Let \( G' = (V, E') \) be the multi-graph.
(If an edge appears in both \( M \) and \( T' \), we count it twice in \( G' \)).
Find an Euler tour \( T' \) of \( G' \).
Output the vertices of \( G \) in the order in which they first appear in \( T' \).
Let \( T \) be the Hamiltonian cycle thus created.

Lemma

The number of odd-degree vertices in \( T' \) is even.

Proof.
Lemma
Let $V' \subseteq V$ such that $|V'|$ is even and let $M$ be a minimum-cost perfect matching on $V'$. Then
\[ c(M) \leq \frac{\text{OPT}(G)}{2} \]

Proof.
?

\qed
Lemma

Let $V' \subseteq V$ such that $|V'|$ is even and let $M$ be a minimum-cost perfect matching on $V'$. Then

$$c(M) \leq \frac{\text{OPT}(G)}{2}$$

Proof.

Let $\mathcal{T}$ be a TSP tour of $G$.

Let $\mathcal{T}'$ be the tour on $V'$ that results by visiting the vertices in $V'$ in the order defined by $\mathcal{T}$.

$$c(\mathcal{T}') \leq c(\mathcal{T}) = \text{OPT}(G)$$

Note that taking every other edge in $\mathcal{T}'$ yields a perfect matching of $V'$ so $\mathcal{T}'$ is the union of two perfect matchings of $V'$, $M'$, and $M''$. Since these matchings cannot have cost less than the minimal one

$$2 \cdot c(M) \leq c(M') + c(M'') = c(\mathcal{T}') \leq \text{OPT}(G).$$

$$c(M) \leq \frac{\text{OPT}(G)}{2}.$$
Theorem

Christofide’s algorithm is $\frac{3}{2}$-approximation for Metric TSP.

Proof.

\[
\begin{align*}
    c(T') & \leq \text{OPT}(G) \\
    c(M) & \leq \frac{\text{OPT}(G)}{2} \\
    c(T') &= c(T') + c(M) \leq \frac{3}{2} \cdot \text{OPT}(G) \\
    c(T) & \leq c(T') \leq \frac{3}{2} \cdot \text{OPT}(G)
\end{align*}
\]
A Negative Result

Theorem

If for any $\rho > 1$, there exists a polynomial-time $\rho$-approximation algorithm for TSP, then there exists a polynomial-time algorithm for solving Hamiltonian cycle (i.e., $P = NP$).

Proof.
Let $A$ be the $\rho$-approximation algorithm for TSP.
Let $G = (V, E)$ be any instance of Hamiltonian cycle, let $G'$ be the complete graph with the same vertex set as $G$ with

$$c_{(u,v)} = \begin{cases} 
1, & (u,v) \in E \\
\rho \cdot |V| + 1, & \text{otherwise}
\end{cases}$$

$G'$ can be constructed in time polynomial in the size of $G$. 
A Negative Result

**Theorem**

If for any $\rho > 1$, there exists a polynomial-time $\rho$-approximation algorithm for TSP, then there exists a polynomial-time algorithm for solving Hamiltonian cycle (i.e., $P = NP$).

**Proof.**

Let $A$ be the $\rho$-approximation algorithm for TSP.

Let $G = (V, E)$ be any instance of Hamiltonian cycle, let $G'$ be the complete graph with the same vertex set as $G$ with

$$c_{(u,v)} = \begin{cases} 
1, & (u,v) \in E \\
\rho \cdot |V| + 1, & \text{otherwise}
\end{cases}$$

$(1)$

$G'$ can be constructed in time polynomial in the size of $G$.

If $G$ has a Hamiltonian cycle $T$, then all edges $e \in T$ have $c_e = 1$ so $T$ is a min-cost tour in $G'$ and $OPT(G') = |V|$. $A(G') \leq \rho \cdot OPT(G') = \rho \cdot |V|$. 
A Negative Result

Theorem

If for any $\rho > 1$, there exists a polynomial-time $\rho$-approximation algorithm for TSP, then there exists a polynomial-time algorithm for solving Hamiltonian cycle (i.e., $P = NP$).

Proof.

Let $A$ be the $\rho$-approximation algorithm for TSP.
Let $G = (V, E)$ be any instance of Hamiltonian cycle, let $G'$ be the complete graph with the same vertex set as $G$ with

$$c(u,v) = \begin{cases} 1, & (u,v) \in E \\ \rho \cdot |V| + 1, & \text{otherwise} \end{cases}$$

$G'$ can be constructed in time polynomial in the size of $G$.
If $G$ has a Hamiltonian cycle $T$, then all edges $e \in T$ have $c_e = 1$ so $T$ is a min-cost tour in $G'$ and $OPT(G') = |V|$. $A(G') \leq \rho \cdot OPT(G') = \rho \cdot |V|$.
If $G$ does not have a Hamiltonian cycle then for every Hamiltonian cycle $T$, $G'$ contains at least one edge $e \notin E$ so $c_e = \rho \cdot |V| + 1$ and

$$c(T) \geq c_e + |V| - 1 = \rho \cdot |V| + 1 + |V| - 1 > \rho \cdot |V|$$

Thus, $A(G') \geq OPT(G') > \rho \cdot |V|$.
A Negative Result

Theorem
A graph $G$ has a Hamiltonian cycle if and only if $A(G') \leq \rho \cdot |V|$.

Proof.
For any $G$,
2. Run $A$ on $G'$ and check if $A(G') \leq \rho \cdot |V|$ or not.
Definition
An instance of the *subset-sum decision problem* is \((S, \ t)\) where
\[ S = \{x_1, \ x_2, \ldots, \ x_n\} \] a set of positive integers and \(t\) a positive integer.
The decision problem is whether some subset of \(S\) adds up exactly to \(t\).
The optimization problem is to find a subset of \(S\) whose sum is as large as possible but no greater than \(t\).

Definition
An *optimization problem* is a family of algorithms \(\{A_\epsilon\}\) such that for each \(\epsilon > 0\), \(A_\epsilon\) is a \((1 - \epsilon)\)-approximation algorithm which runs in polynomial time in input size for fixed \(\epsilon\). \(A_\epsilon\) runs in time polynomial in \(n\) (and \(\frac{1}{\epsilon}\)).
Subset-Sum

1. Get an exact solution.
2. Round/trim input.
3. Get the approximated solution, based on rounded/trimmed input.

Let \( S = \{x_1, x_2, \ldots, x_n\} \). Let \( S + x := \{x_1 + x, x_2 + x, \ldots, x_n + x\} \).

**Algorithm 3.1: Exact-Subset-Sum(\( G \))**

\[
\begin{align*}
n &= |S|; \\
L(0) &= \langle 0 \rangle; \\
\text{for } i = 1 \text{ to } n \\
\{ & L(i) \leftarrow \text{Merge-List}(L(i - 1), L(i - 1) + x_i); \\
& \text{Remove from } L(i) \text{ all elements bigger than } t; \\
\} & \text{Return the largest element in } L(n).
\end{align*}
\]
Trimming

Let \( L = \{x_1, x_2, \ldots, x_m\} \) be a list. To trim the list by parameter \( \delta \) means to remove as many elements from \( L \) as possible in such a way that the list \( L' \) of remaining elements

For every removed \( y \in L \) there exists a \( z \in L' \) such that \((1 - \delta) \cdot y \leq z \leq y\).

**Algorithm 3.2: TRIM\((L, \delta)\)**

\[
L' = < x_1 >; \\
\text{last} = x_1; \\
\text{for } i = 2 \text{ to } m \\
\text{if } \text{last} < (1 - \delta) \cdot x_i \\
\text{append } x_i \text{ onto end of } L'; \\
\text{return } (L)' 
\]
Approximate Subset Sum Problem

**Algorithm 3.3:** APPROXIMATE-SUBSET-SUM($S$, $t$, $\epsilon$)

\[ n = |S|; \]
\[ L(0) = < 0 >; \]
for $i = 1$ to $n$
\[
\begin{align*}
L(i) & = \text{Merge-List}(L(i), L(i - 1) + x_i); \\
L(i) & = \text{trim}(L(i), \epsilon/n); \\
\text{remove from } L(i) \text{ all elements bigger than } t;
\end{align*}
\]
return (max)$L(n)$.

**Proof.**
Let $P_i$ be the set of all values that can be obtained by selecting some subset of $\{x_1, x_2, \ldots, x_i\}$ and summing its members. For every element $y \in P_i$, there exists some $z \in L(i)$ such that $(1 - \epsilon/n)^i \cdot y \leq z \leq y$.

Let $\bar{z}$ be the largest element in $L(n)$. If $y^*$ is a solution to the exact subset-sum problem, then there exists a $z^* \in L(n)$ such that
\[
(1 - \frac{\epsilon}{n})^n \cdot y^* \leq z^* \leq \bar{z} \leq y^*.
\]

\[
\forall n > 1, 1 - \epsilon \leq (1 - \frac{\epsilon}{n})^n, \text{ then } (1 - \epsilon) \cdot y^* \leq \bar{z}.\]
1. Let $x_1, x_2, \ldots, x_n$ be Boolean variables. These variables are set to be either TRUE or FALSE. A variable $x_i$ is TRUE if and only if its negation $\bar{x}_i$ is FALSE and vice versa.

2. A clause is the conjunction of random variables and their negations, e.g., $x_1 \lor \bar{x}_3 \lor x_4$.

3. Given a truth assignment for the $x_1, x_2, \ldots, x_n$, a clause is satisfied if at least one of its elements is TRUE.

4. Given $n$ Boolean variables, $m$ clauses $C_i, \forall i = 1, 2, \ldots, m$ over those variables and a weight $w_i \geq 0$ for each clause, the MAX-SAT problem is to find a truth assignment for the variables that maximizes the total weight of the clauses satisfied. This problem is NP-hard.
Algorithm 4.1: \textsc{MAX-SAT}(n)

\begin{align*}
\text{for } i = 1 \text{ to } n \quad & \left\{ \begin{array}{l}
\text{flip a fair coin.} \\
\text{If “heads” set } x_i \text{ true.} \\
\text{else set } x_i \text{ false.}
\end{array} \right.
\end{align*}

Lemma

Let $\text{OPT}$ be the weight of the optimal assignment and $W$ be the weight of the random assignment. Then

$$E[W] \geq \frac{\text{OPT}}{2}$$

Proof.

?
1. MAX-3SAT is the version of MAX SAT in which every clause $C_j$ has exactly 3 variables in it, i.e., $\forall j, l_j = 3$

2. A theorem due to Hastad says that if there is an approximation algorithm that always returns a solution to the MAX-3SAT that is $\geq \frac{7}{8} \cdot OPT$, then $P = NP$

3. Note that the simple algorithm on the previous page actually returns an assignment whose expectation is $\geq \frac{7}{8} \cdot OPT$ when $\forall j, l_j = 3$. Thus, in some sense, it is a best possible approximation algorithm for MAX-3SAT
A Randomized Approximation Algorithm (Vertex Cover)

An Approximation Algorithm (Metric TSP)

A PTAS (Subset-Sum)

Approximation Algorithms for MAX-3-CNF

A Linear Programming Based (Weighted Vertex Cover)
Weighted Vertex Cover

**Definition**

**Minimum-Weight Vertex Cover Problem.** Given an undirected graph \( G = (V, E) \) in which each vertex \( v \in V \) has an associated positive weight \( w_v \). For any vertex cover \( V' \subseteq V \), we define the weight of the vertex cover \( w(V') = \sum_{v \in V'} w_v \). The goal is to find a vertex cover of minimum weight.

Associate each vertex \( v \) a variable \( x_v \in \{0, 1\} \). We interpret \( x_v = 1 \) as \( v \) is chosen in \( V' \) and \( x_v = 0 \) otherwise. For each edge \((u, v)\), at least one of them is chosen, i.e., \( x_u + x_v \geq 1 \).

\[
\begin{align*}
\text{min} & \quad \sum_{v \in V} w_v \cdot x_v \\
\text{subject to} & \quad x_u + x_v \geq 1, \quad \forall (u, v) \in E \\
& \quad x_v \in \{0, 1\}, \quad \forall v \in V
\end{align*}
\]
Rounding Technique for Integer Programs

\[ \begin{align*}
\min & \quad \sum_{v \in V} w_v \cdot x_v \\
\text{subject to} & \quad x_u + x_v \geq 1, \quad \forall (u, v) \in E \\
& \quad x_v \in \{0, 1\}, \quad \forall v \in V
\end{align*} \]

\[ \begin{align*}
\min & \quad \sum_{v \in V} w_v \cdot x_v \\
\text{subject to} & \quad x_u + x_v \geq 1, \quad \forall (u, v) \in E \\
& \quad x_v \geq 0, \quad \forall v \in V \\
& \quad x_v \leq 1, \quad \forall v \in V
\end{align*} \]
Algorithm 5.1: \textsc{Min-Weight}(G, w)

\[ C = \emptyset; \]
compute \( \bar{x} \), an optimal solution to the linear program;
\textbf{for} each \( v \in V \)
\begin{equation*}
\begin{cases}
\text{if } \bar{x}_v \geq 1/2 \\
\text{then } C = C \cup \{v\};
\end{cases}
\end{equation*}
\textbf{return} \( C \)
Weighted Vertex Cover

Theorem

*Min-Weight is 2-approximation.*

Proof.

Let $C^*$ be an optimal solution to the minimum-weight vertex-cover problem. Let $z^*$ be the value of an optimal solution to the linear program.

\[
\begin{align*}
z^* & \leq w(C^*) \\
z^* & = \sum_{v \in V} w(v) \cdot \bar{x}(v) \\
& \geq \sum_{v \in V: \bar{x}(v) \geq 1/2} w(v) \cdot \bar{x}(v) \geq \sum_{v \in V: \bar{x}(x) \geq 1/2} w(v) \cdot (1/2) \\
& = \sum_{v \in C} w(v) \cdot (1/2) \\
& = (1/2) \cdot \sum_{v \in C} w(v) = (1/2)w(C).
\end{align*}
\]