Approximation and Online Algorithms: CS60023: Autumn 2018: S P Pal *Copyrights reserved*

Instructor: Sudebkumar Prasant Pal

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Heuristics for vertex cover for graph $G(E, V)$

- Select (and delete) an arbitrary vertex $v \in V$ for inclusion in vertex cover C and drop all edges incident on v. Repeat.

Heuristics for vertex cover for graph $G(E,V)$

- Select (and delete) an arbitrary vertex $v \in V$ for inclusion in vertex cover C and drop all edges incident on v. Repeat.
- Select an arbitrary edge $\{u,v\} \in E$ and include u and v in C. Drop all edges incident on u and v. Repeat.

Heuristics for vertex cover for graph $G(E, V)$

- Select (and delete) an arbitrary vertex $v \in V$ for inclusion in vertex cover C and drop all edges incident on v. Repeat.
- Select an arbitrary edge $\{u, v\} \in E$ and include u and v in C. Drop all edges incident on u and v. Repeat.
- For a straight line graph, the first method find a vertex cover of size $n - 1$, within twice the optimal. Does it work well also for other classes of graphs like trees, planar graphs, and general graphs?

Heuristics for vertex cover for graph $G(E, V)$

- Select (and delete) an arbitrary vertex $v \in V$ for inclusion in vertex cover C and drop all edges incident on v. Repeat.
- Select an arbitrary edge $\{u, v\} \in E$ and include u and v in C. Drop all edges incident on u and v. Repeat.
- For a straight line graph, the first method find a vertex cover of size $n - 1$, within twice the optimal. Does it work well also for other classes of graphs like trees, planar graphs, and general graphs?
- The second one chooses both vertices of all edges in a maximal matching S.

The factor two ratio bound

- Let S be any maximal matching. Our computed vertex cover has the $2|S|$ vertices defining these edges of S.

The factor two ratio bound

- Let S be any maximal matching. Our computed vertex cover has the $2|S|$ vertices defining these edges of S.
- These vertices for a vertex cover because any uncovered edge would have given rise to another matching edge contradicting the maximality of S.

The factor two ratio bound

- Let S be any maximal matching. Our computed vertex cover has the $2|S|$ vertices defining these edges of S.
- These vertices for a vertex cover because any uncovered edge would have given rise to another matching edge contradicting the maximality of S.
- If C^* is any minimum vertex cover then $|S| \leq |C^*|$. Why?

The factor two ratio bound

- Let S be any maximal matching. Our computed vertex cover has the $2|S|$ vertices defining these edges of S.
- These vertices for a vertex cover because any uncovered edge would have given rise to another matching edge contradicting the maximality of S.
- If C^* is any minimum vertex cover then $|S| \leq |C^*|$. Why?
- Vertices in $C*$ have to cover each edge in the matching S. So, C^* must include at least one vertex from each of the S edges.

The factor two ratio bound

- Let S be any maximal matching. Our computed vertex cover has the $2|S|$ vertices defining these edges of S.
- These vertices for a vertex cover because any uncovered edge would have given rise to another matching edge contradicting the maximality of S.
- If C^* is any minimum vertex cover then $|S| \leq |C^*|$. Why?
- Vertices in $C*$ have to cover each edge in the matching S. So, C^* must include at least one vertex from each of the S edges.
- The $2|S|$ vertices comprise C, the computed approximate vertex cover, where $|C^*| \leq |C| = 2|S| \leq 2|C^*|$.

The factor two ratio bound

- Let S be any maximal matching. Our computed vertex cover has the $2|S|$ vertices defining these edges of S.
- These vertices for a vertex cover because any uncovered edge would have given rise to another matching edge contradicting the maximality of S.
- If C^* is any minimum vertex cover then $|S| \leq |C^*|$. Why?
- Vertices in $C*$ have to cover each edge in the matching S. So, C^* must include at least one vertex from each of the S edges.
- The $2|S|$ vertices comprise C, the computed approximate vertex cover, where $|C^*| \leq |C| = 2|S| \leq 2|C^*|$.
- This yields a vertex cover that is certainly at most twice the size of the minimum vertex cover.

The NP-completeness reduction

- The NP-complete reduction from 3-SAT to vertex cover constructs a graph $G_f(V_f, E_f)$ for each 3-SAT CNF formula f, such that f is satisfiable if and only if G_f has a vertex cover of size exactly $k = 2|V_f|/3$.

The NP-completeness reduction

- The NP-complete reduction from 3-SAT to vertex cover constructs a graph $G_f(V_f, E_f)$ for each 3-SAT CNF formula f, such that f is satisfiable if and only if G_f has a vertex cover of size exactly $k = 2|V_f|/3$.
- Here V_f is the set of vertices, one for each of the $3m$ literals from the m clauses in f.

The NP-completeness reduction

- The NP-complete reduction from 3-SAT to vertex cover constructs a graph $G_f(V_f, E_f)$ for each 3-SAT CNF formula f, such that f is satisfiable if and only if G_f has a vertex cover of size exactly $k = 2|V_f|/3$.
- Here V_f is the set of vertices, one for each of the $3m$ literals from the m clauses in f.
- The edges form triangles for each clause; three edges between pairs of literals in each clause.

The NP-completeness reduction

- The NP-complete reduction from 3-SAT to vertex cover constructs a graph $G_f(V_f, E_f)$ for each 3-SAT CNF formula f, such that f is satisfiable if and only if G_f has a vertex cover of size exactly $k = 2|V_f|/3$.
- Here V_f is the set of vertices, one for each of the $3m$ literals from the m clauses in f.
- The edges form triangles for each clause; three edges between pairs of literals in each clause.
- More edges join inconsistent pairs of literals across the clause triangles, like x_i with x_i'.

The NP-completeness reduction

- The NP-complete reduction from 3-SAT to vertex cover constructs a graph $G_f(V_f, E_f)$ for each 3-SAT CNF formula f, such that f is satisfiable if and only if G_f has a vertex cover of size exactly $k = 2|V_f|/3$.
- Here V_f is the set of vertices, one for each of the $3m$ literals from the m clauses in f.
- The edges form triangles for each clause; three edges between pairs of literals in each clause.
- More edges join inconsistent pairs of literals across the clause triangles, like x_i with x_i'.
- Note that the minimum vertex cover must have size at least $2/3|V_f|$.

The NP-completeness reduction

- Suppose f is a satisfiable formula. We show that every minimum vertex cover of G_f will have exactly $2/3|V_f| = 2m$ vertices.

The NP-completeness reduction

- Suppose f is a satisfiable formula. We show that every minimum vertex cover of G_f will have exactly $2/3|V_f| = 2m$ vertices.
- In other words, we show that $2m$ vertices are both necessary and sufficient for making the minimum cardinality vertex cover; we need at least $2m$ vertices to cover all the $3m$ edges of m triangles.

The NP-completeness reduction

- Suppose f is a satisfiable formula. We show that every minimum vertex cover of G_f will have exactly $2/3|V_f| = 2m$ vertices.
- In other words, we show that $2m$ vertices are both necessary and sufficient for making the minimum cardinality vertex cover; we need at least $2m$ vertices to cover all the $3m$ edges of m triangles.
- Let us consider any truth assignment U for f.

The NP-completeness reduction

- Suppose f is a satisfiable formula. We show that every minimum vertex cover of G_f will have exactly $2/3|V_f| = 2m$ vertices.
- In other words, we show that $2m$ vertices are both necessary and sufficient for making the minimum cardinality vertex cover; we need at least $2m$ vertices to cover all the $3m$ edges of m triangles.
- Let us consider any truth assignment U for f.
- The function U assigns a value 'T' or 'F' to each variable of f, thereby assigning true or false value to each literal in each clause.

The NP-completeness reduction

- Suppose f is a satisfiable formula. We show that every minimum vertex cover of G_f will have exactly $2/3|V_f| = 2m$ vertices.
- In other words, we show that $2m$ vertices are both necessary and sufficient for making the minimum cardinality vertex cover; we need at least $2m$ vertices to cover all the $3m$ edges of m triangles.
- Let us consider any truth assignment U for f.
- The function U assigns a value 'T' or 'F' to each variable of f, thereby assigning true or false value to each literal in each clause.
- In G_f, we have one vertex for each literal of each clause, totalling to $3m$ vertices in all.

The NP-completeness reduction

- We know that U falsifies at most two literals in each clause, leaving at least one literal in each clause satisfied.

The NP-completeness reduction

- We know that U falsifies at most two literals in each clause, leaving at least one literal in each clause satisfied.
- Consider the cross edges in G_f that run between the triangles.

The NP-completeness reduction

- We know that U falsifies at most two literals in each clause, leaving at least one literal in each clause satisfied.
- Consider the cross edges in G_f that run between the triangles.
- We must choose two vertices from each triangle so that all these cross edges are also covered; two vertices per triangle will in any case cover the edges of the triangles.

The NP-completeness reduction

- We know that U falsifies at most two literals in each clause, leaving at least one literal in each clause satisfied.
- Consider the cross edges in G_f that run between the triangles.
- We must choose two vertices from each triangle so that all these cross edges are also covered; two vertices per triangle will in any case cover the edges of the triangles.
- Since each cross edge can be covered from either end, we choose the end vertex whose corresponding literal is falsified by the truth assignment U.

The NP-completeness reduction

- We know that U falsifies at most two literals in each clause, leaving at least one literal in each clause satisfied.
- Consider the cross edges in G_f that run between the triangles.
- We must choose two vertices from each triangle so that all these cross edges are also covered; two vertices per triangle will in any case cover the edges of the triangles.
- Since each cross edge can be covered from either end, we choose the end vertex whose corresponding literal is falsified by the truth assignment U.

The NP-completeness reduction

- We thus cover all cross edges by choosing at most two vertices per triangle because each clause corresponding to any triangle has at most two falsified literals.

The NP-completeness reduction

- We thus cover all cross edges by choosing at most two vertices per triangle because each clause corresponding to any triangle has at most two falsified literals.
- If we have selected no vertex (or only a single vertex) from some triangle then we simply select two (or one) additional vertex from the traingle arbitrarily, so that a total of two vertices are selected from each triangle.

The NP-completeness reduction

- We thus cover all cross edges by choosing at most two vertices per triangle because each clause corresponding to any triangle has at most two falsified literals.
- If we have selected no vertex (or only a single vertex) from some triangle then we simply select two (or one) additional vertex from the traingle arbitrarily, so that a total of two vertices are selected from each triangle.
- This shows the existence of a vertex cover of minimal cardinality $2m$ in G_f.

The NP-completeness reduction

- We thus cover all cross edges by choosing at most two vertices per triangle because each clause corresponding to any triangle has at most two falsified literals.
- If we have selected no vertex (or only a single vertex) from some triangle then we simply select two (or one) additional vertex from the traingle arbitrarily, so that a total of two vertices are selected from each triangle.
- This shows the existence of a vertex cover of minimal cardinality $2m$ in G_f.

The NP-completeness reduction

- We thus cover all cross edges by choosing at most two vertices per triangle because each clause corresponding to any triangle has at most two falsified literals.
- If we have selected no vertex (or only a single vertex) from some triangle then we simply select two (or one) additional vertex from the traingle arbitrarily, so that a total of two vertices are selected from each triangle.
- This shows the existence of a vertex cover of minimal cardinality $2m$ in G_f.

The NP-completeness reduction

- Conversely, suppose G_f has a minimum vertex cover C of size exactly $2/3|V_f| = 2m$.

The NP-completeness reduction

- Conversely, suppose G_f has a minimum vertex cover C of size exactly $2/3|V_f| = 2m$.
- We show that f must be satisfiable; we generate a truth assignment U that satisfies at least one literal in each clause.

The NP-completeness reduction

- Conversely, suppose G_f has a minimum vertex cover C of size exactly $2/3|V_f| = 2m$.
- We show that f must be satisfiable; we generate a truth assignment U that satisfies at least one literal in each clause.
- The vertex cover C has exactly two vertices in each triangle and certainly covers each cross edge.

The NP-completeness reduction

- Conversely, suppose G_f has a minimum vertex cover C of size exactly $2/3|V_f| = 2m$.
- We show that f must be satisfiable; we generate a truth assignment U that satisfies at least one literal in each clause.
- The vertex cover C has exactly two vertices in each triangle and certainly covers each cross edge.
- So, any cross edge e is covered at least at one of its ends by some vertex in C.

The NP-completeness reduction

- Conversely, suppose G_f has a minimum vertex cover C of size exactly $2/3|V_f| = 2m$.
- We show that f must be satisfiable; we generate a truth assignment U that satisfies at least one literal in each clause.
- The vertex cover C has exactly two vertices in each triangle and certainly covers each cross edge.
- So, any cross edge e is covered at least at one of its ends by some vertex in C.
- If both ends are covered then we choose any one, say vertex v arbitrarily.

The NP-completeness reduction

- If $x(v)$ is the boolean variable in the 3-CNF formula corresponding to the vertex v, then we assign $x(v) = f$ if the literal corresponding to the vertex v is the uncomplemented literal for boolean variable $x(v)$, and we assign $x(v) = t$, otherwise.

The NP-completeness reduction

- If $x(v)$ is the boolean variable in the 3-CNF formula corresponding to the vertex v, then we assign $x(v) = f$ if the literal corresponding to the vertex v is the uncomplemented literal for boolean variable $x(v)$, and we assign $x(v) = t$, otherwise.
- So, the literal at the other end of the cross edge e is assigned ‘T’ in the truth assignment with $x(v)$ assigned as above.

The NP-completeness reduction

- If $x(v)$ is the boolean variable in the 3-CNF formula corresponding to the vertex v, then we assign $x(v) = f$ if the literal corresponding to the vertex v is the uncomplemented literal for boolean variable $x(v)$, and we assign $x(v) = t$, otherwise.
- So, the literal at the other end of the cross edge e is assigned 'T' in the truth assignment with $x(v)$ assigned as above.
- In this way, the truth value 'F' is assigned for exactly two literals of the formula in every clause, corrresponding to the two vertices of the vertex cover C of the corresponding triangle.

The NP-completeness reduction

- If the literal of the third vertex in any triangle is not assigned any truth value in this manner, then we know that this literal does not appear complimented in any other clause; we can therefore do truth assignment to its boolean variable accordingly, so that this literal is satisfied, thereby satisfying the clause.

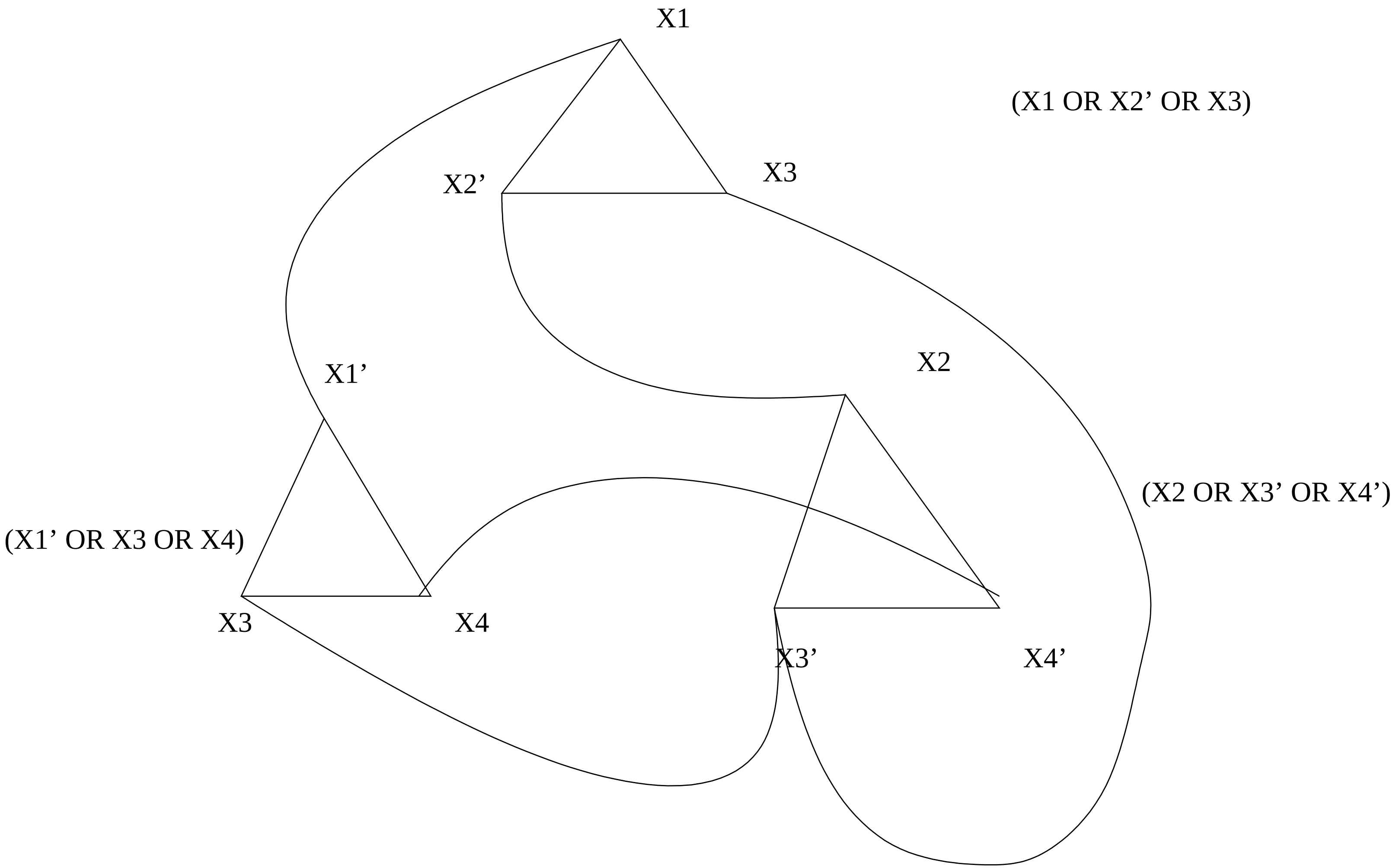
The NP-completeness reduction

- If the literal of the third vertex in any triangle is not assigned any truth value in this manner, then we know that this literal does not appear complimented in any other clause; we can therefore do truth assignment to its boolean variable accordingly, so that this literal is satisfied, thereby satisfying the clause.
- This completes the truth assignment for every literal in the formula where no clause has more than two literals falsified.

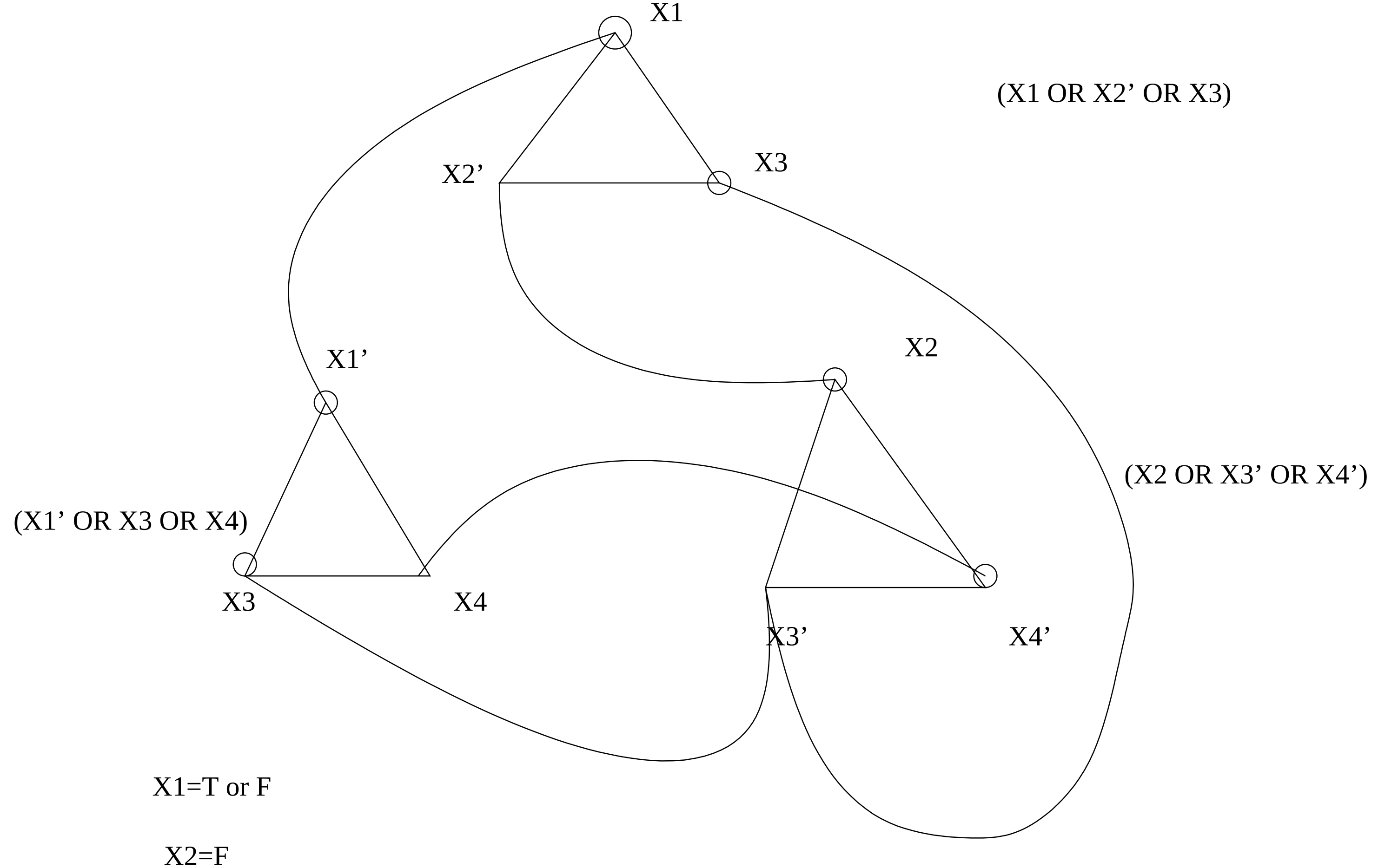
The NP-completeness reduction

- If the literal of the third vertex in any triangle is not assigned any truth value in this manner, then we know that this literal does not appear complimented in any other clause; we can therefore do truth assignment to its boolean variable accordingly, so that this literal is satisfied, thereby satisfying the clause.
- This completes the truth assignment for every literal in the formula where no clause has more than two literals falsified.
- Therefore the truth assignment thus designed must be a satisfying truth assignment for the boolean 3-CNF formula.

The construction of G_f from 3-CNF formula f



One minimum vertex cover

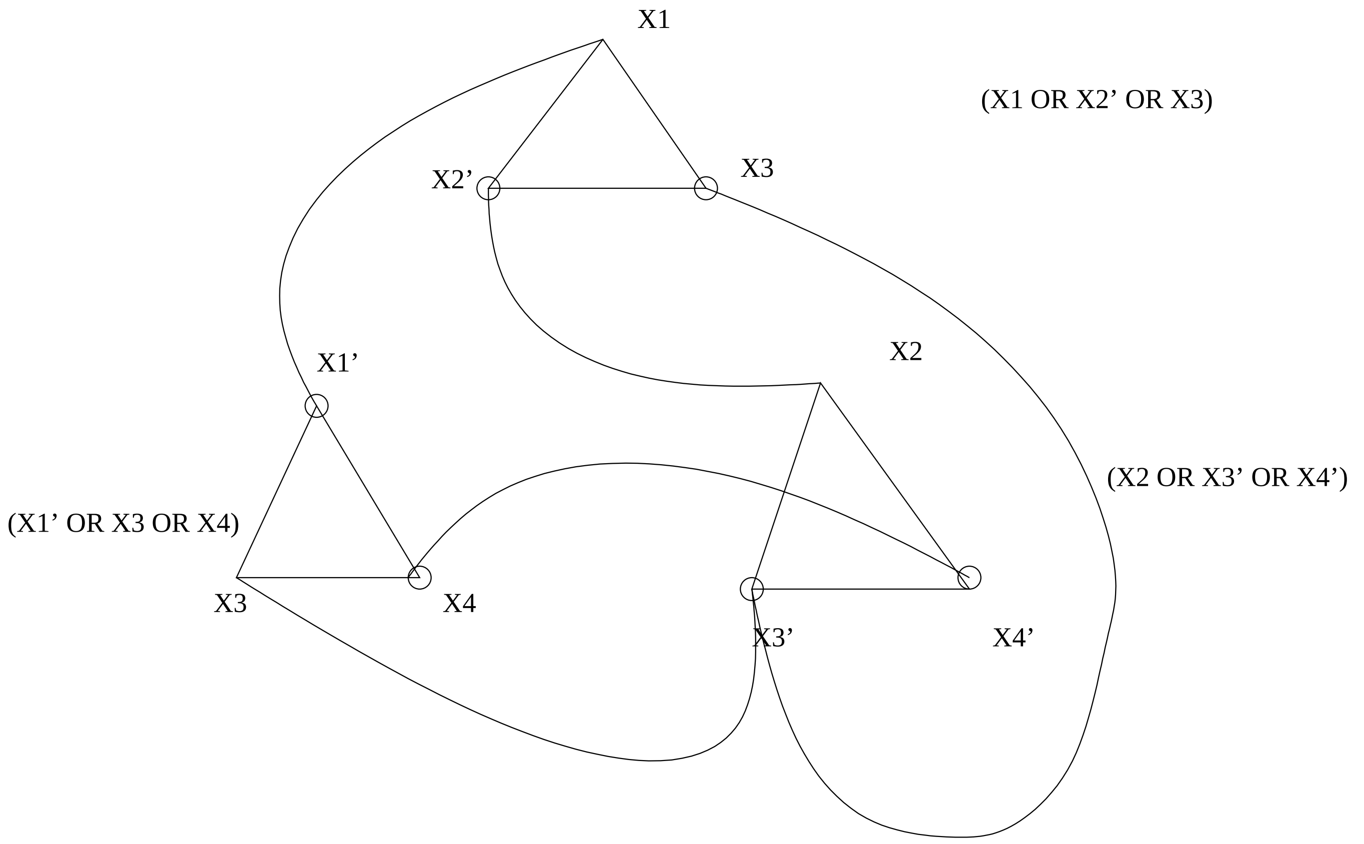


X1=T or F

X2=F

X3=F X4=T

Another minimum vertex cover



X1=T

X4=T or F

X2=T

X3=T

The Art Gallery Problem

- Greedy triangulation of a simple polygon of n vertices into triangles with vertices coinciding with polygonal vertices.

The Art Gallery Problem

- Greedy triangulation of a simple polygon of n vertices into triangles with vertices coinciding with polygonal vertices.
- The existence of several triangulations of a simple polygon or a convex polygon; the greedy algorithm has choices at each stage.

The Art Gallery Problem

- Greedy triangulation of a simple polygon of n vertices into triangles with vertices coinciding with polygonal vertices.
- The existence of several triangulations of a simple polygon or a convex polygon; the greedy algorithm has choices at each stage.
- There may be an exponential number of triangulations.

The Art Gallery Problem

- Greedy triangulation of a simple polygon of n vertices into triangles with vertices coinciding with polygonal vertices.
- The existence of several triangulations of a simple polygon or a convex polygon; the greedy algorithm has choices at each stage.
- There may be an exponential number of triangulations.
- There is a greedy 3-colouring of the vertices of the polygon with respect to the triangulation graph.

The Art Gallery Problem

- Greedy triangulation of a simple polygon of n vertices into triangles with vertices coinciding with polygonal vertices.
- The existence of several triangulations of a simple polygon or a convex polygon; the greedy algorithm has choices at each stage.
- There may be an exponential number of triangulations.
- There is a greedy 3-colouring of the vertices of the polygon with respect to the triangulation graph.
- The vertices getting the colour which colours the smallest number of vertices are at most $\lfloor n/3 \rfloor$ in number.

The Art Gallery Problem

- They cover all triangles because each triangle has a vertex with each of the three colours.

The Art Gallery Problem

- They cover all triangles because each triangle has a vertex with each of the three colours.
- The similarity with vertex cover for graphs is that vertices covered edges for graphs, whereas vertices cover regions (triangles) of the polygon for the art gallery problem.

The Art Gallery Problem

- They cover all triangles because each triangle has a vertex with each of the three colours.
- The similarity with vertex cover for graphs is that vertices covered edges for graphs, whereas vertices cover regions (triangles) of the polygon for the art gallery problem.
- Savings in the number of vertex guards is possible if we note that several guards see common regions, beyond their own triangles. The art gallery problem of minimizing vertex guards is NP-hard.

The general set cover problem

- We now generalize the vertex cover problem to general sets of objects from a universal set U.

The general set cover problem

- We now generalize the vertex cover problem to general sets of objects from a universal set U.
- Let $\mathcal{S}$ be a collection of k subsets $S \subseteq U$ such that $\mathcal{S}$ covers U, that is $\cup_{S \in \mathcal{S}}(S) = U$.

The general set cover problem

- We now generalize the vertex cover problem to general sets of objects from a universal set U.
- Let $\mathcal{S}$ be a collection of k subsets $S \subseteq U$ such that $\mathcal{S}$ covers U, that is $\cup_{S \in \mathcal{S}}(S) = U$.
- We wish to find the smallest cardinality collection $C^* \subseteq \mathcal{S}$ of sets so that C^* covers U, that is $\cup_{S \in C^*}(S) = U$.

The general set cover problem

- We now generalize the vertex cover problem to general sets of objects from a universal set U.
- Let $\mathcal{S}$ be a collection of k subsets $S \subseteq U$ such that $\mathcal{S}$ covers U, that is $\cup_{S \in \mathcal{S}}(S) = U$.
- We wish to find the smallest cardinality collection $C^* \subseteq \mathcal{S}$ of sets so that C^* covers U, that is $\cup_{S \in C^*}(S) = U$.
- In order to view this as a 'vertex covering' problem, consider the set $\mathcal{S}$ of k sets as corresponding to the set V of all vertices in the graph $G(V, E)$, where the set E has one element $\mathcal{S}_u \in E$ for each element $u \in U$ and each set $S \subseteq U$ such that $u \in S \in \mathcal{S}_u$, and such that $\mathcal{S}_u \subseteq \mathcal{S}$.

The general set cover problem

- We now generalize the vertex cover problem to general sets of objects from a universal set U.
- Let $\mathcal{S}$ be a collection of k subsets $S \subseteq U$ such that $\mathcal{S}$ covers U, that is $\cup_{S \in \mathcal{S}}(S) = U$.
- We wish to find the smallest cardinality collection $C^* \subseteq \mathcal{S}$ of sets so that C^* covers U, that is $\cup_{S \in C^*}(S) = U$.
- In order to view this as a 'vertex covering' problem, consider the set $\mathcal{S}$ of k sets as corresponding to the set V of all vertices in the graph $G(V, E)$, where the set E has one element $\mathcal{S}_u \in E$ for each element $u \in U$ and each set $S \subseteq U$ such that $u \in S \in \mathcal{S}_u$, and such that $\mathcal{S}_u \subseteq \mathcal{S}$.
- The n elements in U correspond to the (hyper)edges $\mathcal{S}_u$ of the (hyper)graph $G(V, E)$.

The general set cover problem

- The vertex cover problem is a special case of the set cover problem.

The general set cover problem

- The vertex cover problem is a special case of the set cover problem.
- For the general set cover problem, we wish to find a good (small) cover $C \subseteq \mathcal{S}$ of U such that $\cup_{S \in C}(S) = U$.

The general set cover problem

- The vertex cover problem is a special case of the set cover problem.
- For the general set cover problem, we wish to find a good (small) cover $C \subseteq \mathcal{S}$ of U such that $\cup_{S \in C}(S) = U$.
- We show that such a cover C can be found in polynomial time with ratio bound $O(\log |U|)$, that is, $|C| = O(|C^*| \log |U|)$.

The general set cover problem

- The vertex cover problem is a special case of the set cover problem.
- For the general set cover problem, we wish to find a good (small) cover $C \subseteq \mathcal{S}$ of U such that $\cup_{S \in C}(S) = U$.
- We show that such a cover C can be found in polynomial time with ratio bound $O(\log |U|)$, that is, $|C| = O(|C^*| \log |U|)$.
- Surprisingly, a simple heuristic works; we choose sets $S \in \mathcal{S}$ in decreasing order of the number of new elements covered, until all elements of U are covered. The sets thus selected constitute the collection $C \subseteq \mathcal{S}$.

The logarithmic approximation ratio

- In order to establish the approximation ratio, we need a *charging scheme* for already covered elements.

The logarithmic approximation ratio

- In order to establish the approximation ratio, we need a *charging scheme* for already covered elements.
- We add one set at a time to the set cover. Whenever we select the next set to be included in the the set cover in our approximation algorithm, we assign some ‘prices’ to (only) the new elements of the new set as follows.

The logarithmic approximation ratio

- In order to establish the approximation ratio, we need a *charging scheme* for already covered elements.
- We add one set at a time to the set cover. Whenever we select the next set to be included in the the set cover in our approximation algorithm, we assign some ‘prices’ to (only) the new elements of the new set as follows.
- If the $i - 1$ sets selected so far are S_1, S_2, ...,S_{i-1}, then we have already assigned some ‘prices’ to the elements of these sets.

The logarithmic approximation ratio

- When the subsequent set S_i is selected, some more elements from S_i are introduced in to the set cover that were not covered by the previous $i - 1$ sets.

The logarithmic approximation ratio

- When the subsequent set S_i is selected, some more elements from S_i are introduced in to the set cover that were not covered by the previous $i-1$ sets.
- The set S_i is selected because it has the largest number $|S_i \setminus (S_1 \cup S_2 \cup \cdots \cup S_{i-1})|$ of new elements amongst all the sets in $\mathcal{S} \setminus \{S_1, S_2, \cdots, S_{i-1}\}$.

The logarithmic approximation ratio

- When the subsequent set S_i is selected, some more elements from S_i are introduced in to the set cover that were not covered by the previous $i-1$ sets.
- The set S_i is selected because it has the largest number $|S_i \setminus (S_1 \cup S_2 \cup \cdots \cup S_{i-1})|$ of new elements amongst all the sets in $\mathcal{S} \setminus \{S_1, S_2, \cdots, S_{i-1}\}$.
- The *price* applied on each new element is $\frac{1}{|S_i \setminus (S_1 \cup S_2 \cup \cdots S_{i-1})|}$.

The logarithmic approximation ratio

- When the subsequent set S_i is selected, some more elements from S_i are introduced in to the set cover that were not covered by the previous $i-1$ sets.
- The set S_i is selected because it has the largest number $|S_i \setminus (S_1 \cup S_2 \cup \cdots \cup S_{i-1})|$ of new elements amongst all the sets in $\mathcal{S} \setminus \{S_1, S_2, \cdots, S_{i-1}\}$.
- The *price* applied on each new element is $\frac{1}{|S_i \setminus (S_1 \cup S_2 \cup \cdots S_{i-1})|}$.
- Each element is charged with a *price* only once; let the *price* assigned to an element $x \in U$ be c_x.

The upper bound

- Observe that the sum of all weights charged is $|\mathcal{C}| = \sum_{x \in U} c_x$.

The upper bound

- Observe that the sum of all weights charged is $|\mathcal{C}| = \sum_{x \in U} c_x$.
- Each new element is charged (only once) with a price that is the inverse of the number of *new elements* introduced by the new set containing them. So, the sum total of all weights is equal to the number $|\mathcal{C}|$, of sets selected by the approximation algorithm.

The upper bound

- Observe that the sum of all weights charged is $|\mathcal{C}| = \sum_{x \in U} c_x$.
- Each new element is charged (only once) with a price that is the inverse of the number of *new elements* introduced by the new set containing them. So, the sum total of all weights is equal to the number $|\mathcal{C}|$, of sets selected by the approximation algorithm.
- We now define a quantity $\sum_{S \in \mathcal{C}^*} \sum_{x \in S} c_x$ for an (unknown) optimal set cover $\mathcal{C}^*$. [We will succeed in showing that this quantity is indeed an upper bound on $|\mathcal{C}|$.]

The sandwiching upper bound

- Observe that $|\mathcal{C}| = \sum_{x \in X} c_x \leq \sum_{S \in \mathcal{C}^*} \sum_{x \in S} c_x$

The sandwiching upper bound

- Observe that $|\mathcal{C}| = \sum_{x \in X} c_x \leq \sum_{S \in \mathcal{C}^*} \sum_{x \in S} c_x$
- The reason is that the sets in the optimal cover $\mathcal{C}^*$ might intersect. So, an element of x may be counted several times on the right hand side of the inequality whereas each element is counted exactly once on the left hand side.

The sandwiching upper bound

- Observe that $|\mathcal{C}| = \sum_{x \in X} c_x \leq \sum_{S \in \mathcal{C}^*} \sum_{x \in S} c_x$
- The reason is that the sets in the optimal cover $\mathcal{C}^*$ might intersect. So, an element of x may be counted several times on the right hand side of the inequality whereas each element is counted exactly once on the left hand side.
- We assume (as shown in Cormen et al. pages 1036-37), that $\sum_{x \in S} c_x \leq H(|S|) | S \in \mathcal{S}$

The sandwiching upper bound

- Observe that $|\mathcal{C}| = \sum_{x \in X} c_x \leq \sum_{S \in \mathcal{C}^*} \sum_{x \in S} c_x$
- The reason is that the sets in the optimal cover $\mathcal{C}^*$ might intersect. So, an element of x may be counted several times on the right hand side of the inequality whereas each element is counted exactly once on the left hand side.
- We assume (as shown in Cormen et al. pages 1036-37), that $\sum_{x \in S} c_x \leq H(|S|)|S \in \mathcal{S}$
- Here, $H(n) = O(\log n)$ is the harmonic sum $\sum_{1 \leq i \leq n} \frac{1}{i}$

The sandwiching upper bound

- Observe that $|\mathcal{C}| = \sum_{x \in X} c_x \leq \sum_{S \in \mathcal{C}^*} \sum_{x \in S} c_x$
- The reason is that the sets in the optimal cover $\mathcal{C}^*$ might intersect. So, an element of x may be counted several times on the right hand side of the inequality whereas each element is counted exactly once on the left hand side.
- We assume (as shown in Cormen et al. pages 1036-37), that $\sum_{x \in S} c_x \leq H(|S|)|S \in \mathcal{S}$
- Here, $H(n) = O(\log n)$ is the harmonic sum $\sum_{1 \leq i \leq n} \frac{1}{i}$
- We can now see that $|\mathcal{C}| \leq \sum_{S \in \mathcal{C}^*} H(|S|) \leq |\mathcal{C}^*|.H(max\{|S| : S \in \mathcal{S}\})$

The general weighted set cover problem

- In this case, each set $S \subseteq U$, has a positive and rational weight $c(S)$. Here, U is the universal set of n elements and the collection of sets $\mathcal{S} = \{S_1, S_2, \cdots, S_k\}$.

The general weighted set cover problem

- In this case, each set $S \subseteq U$, has a positive and rational weight $c(S)$. Here, U is the universal set of n elements and the collection of sets $\mathcal{S} = \{S_1, S_2, \cdots, S_k\}$.
- We need to find the weighted minimum subset of $\mathcal{S}$, that covers U, given that $\mathcal{S}$ covers U.

The general weighted set cover problem

- In this case, each set $S \subseteq U$, has a positive and rational weight $c(S)$. Here, U is the universal set of n elements and the collection of sets $\mathcal{S} = \{S_1, S_2, \cdots, S_k\}$.
- We need to find the weighted minimum subset of $\mathcal{S}$, that covers U, given that $\mathcal{S}$ covers U.
- We now show how the ratio approximation factor of $H(n)$ is attained.

The general weighted set cover problem

- In this case, each set $S \subseteq U$, has a positive and rational weight $c(S)$. Here, U is the universal set of n elements and the collection of sets $\mathcal{S} = \{S_1, S_2, \cdots, S_k\}$.
- We need to find the weighted minimum subset of $\mathcal{S}$, that covers U, given that $\mathcal{S}$ covers U.
- We now show how the ratio approximation factor of $H(n)$ is attained.
- The greedy selection rule for the next set S is similar to the rule in the unweighted set cover heuristic.

The general weighted set cover problem

- If the set of already covered elements is C, then $|S \setminus C|$ is the number of new elements.

The general weighted set cover problem

- If the set of already covered elements is C, then $|S \setminus C|$ is the number of new elements.
- Let $c(S)$ be the cost of S. Then the cost per element added afresh is $\alpha = \frac{c(S)}{|S \setminus C|}$.

The general weighted set cover problem

- If the set of already covered elements is C, then $|S \setminus C|$ is the number of new elements.
- Let $c(S)$ be the cost of S. Then the cost per element added afresh is $\alpha = \frac{c(S)}{|S \setminus C|}$.
- This is called the *cost effectiveness* of the set S.

The general weighted set cover problem

- If the set of already covered elements is C, then $|S \setminus C|$ is the number of new elements.
- Let $c(S)$ be the cost of S. Then the cost per element added afresh is $\alpha = \frac{c(S)}{|S \setminus C|}$.
- This is called the *cost effectiveness* of the set S.
- In each greedy step, we select that set S whose cost effectiveness is minimum; for each element $e \in S$, we say $price(e) = \alpha$.

The general weighted set cover problem

- If the set of already covered elements is C, then $|S \setminus C|$ is the number of new elements.
- Let $c(S)$ be the cost of S. Then the cost per element added afresh is $\alpha = \frac{c(S)}{|S \setminus C|}$.
- This is called the *cost effectiveness* of the set S.
- In each greedy step, we select that set S whose cost effectiveness is minimum; for each element $e \in S$, we say $price(e) = \alpha$.
- Now, let $e_1, e_2, \cdots, e_n$ be the sequence in which the selected sets covered the n elements.

The general weighted set cover problem

- We have the following non-trivial upper bound:

$$price(e_k) \leq \frac{OPT}{n-k+1}$$

The general weighted set cover problem

- We have the following non-trivial upper bound:

$$price(e_k) \leq \frac{OPT}{n-k+1}$$

- We establish this bound below; first we show how to use this bound.

The general weighted set cover problem

- We have the following non-trivial upper bound:

$$price(e_k) \leq \frac{OPT}{n-k+1}$$

- We establish this bound below; first we show how to use this bound.
- We know that summing $price(e)$ over all $e \in U$ gives us the sum of weights of sets in the set cover computed by our greedy algorithm.

The general weighted set cover problem

- We have the following non-trivial upper bound:

$$price(e_k) \leq \frac{OPT}{n-k+1}$$

- We establish this bound below; first we show how to use this bound.
- We know that summing $price(e)$ over all $e \in U$ gives us the sum of weights of sets in the set cover computed by our greedy algorithm.
- This is clearly $H(n) \times OPT$, by the use of the above upper bound for $price(e)$, which we now proceed to establish below.

The general weighted set cover problem

- We now establish the upper bound on $price(e)$.

The general weighted set cover problem

- We now establish the upper bound on $price(e)$.
- At any stage of our greedy algorithm, consider the leftover sets T_1, T_2, ..., T_p of the optimal solution and the remaining elements to be covered.

The general weighted set cover problem

- We now establish the upper bound on $price(e)$.
- At any stage of our greedy algorithm, consider the leftover sets T_1, T_2, ..., T_p of the optimal solution and the remaining elements to be covered.
- Since all the n elements can be covered with OPT cost by the already chosen sets of the optimal solution C^* and the leftover sets of the optimal solution, it is also possible for the leftover sets of the optimal solution to cover the remaining elements with cost at most OPT.

The general weighted set cover problem

- We now establish the upper bound on $price(e)$.
- At any stage of our greedy algorithm, consider the leftover sets T_1, T_2, ..., T_p of the optimal solution and the remaining elements to be covered.
- Since all the n elements can be covered with OPT cost by the already chosen sets of the optimal solution C^* and the leftover sets of the optimal solution, it is also possible for the leftover sets of the optimal solution to cover the remaining elements with cost at most OPT.
- The sum of costs of all remaining sets of the optimal cover is no more than OPT.

The general weighted set cover problem

- The number of elements remaining to be covered is $|U \setminus C|$.

The general weighted set cover problem

- The number of elements remaining to be covered is $|U \setminus C|$.
- The cost of covering these elements is no more than OPT.

The general weighted set cover problem

- The number of elements remaining to be covered is $|U \setminus C|$.
- The cost of covering these elements is no more than OPT.
- There must be one such set with cost effectiveness at most the upper bound of
$$\frac{OPT}{|U \setminus C|}$$
. Why?

The general weighted set cover problem

- The number of elements remaining to be covered is $|U \setminus C|$.
- The cost of covering these elements is no more than OPT.
- There must be one such set with cost effectiveness at most the upper bound of

$$\frac{OPT}{|U \setminus C|}$$

. Why?

- Therefore, our algorithm will greedily select some set covering the kth element with at most

$$price(e_k) \leq \frac{OPT}{|U \setminus C|} \leq \frac{OPT}{n-k+1}$$

Linear programming duality

- The linear program with m *linear inequalities* representing *constraints* for minimizing a *linear objective function* for an n-dimensional *non-negative vector* is as follows.

Linear programming duality

- The linear program with m *linear inequalities* representing *constraints* for minimizing a *linear objective function* for an n-dimensional *non-negative vector* is as follows.
- $minimize \sum_{j=1}^{n} c_j x_j [c^T x]$
  $given \sum_{j=1}^{n} a_{ij} x_j \geq b_i, i = 1, \cdots, m [Ax \geq b]$
  $x_j \geq 0, j = 1, \cdots, n [x \geq 0]$ where a_{ij}, b_i, c_j are given rational numbers.

Linear programming duality

- The linear program with m *linear inequalities* representing *constraints* for minimizing a *linear objective function* for an n-dimensional *non-negative vector* is as follows.
- $minimize \sum_{j=1}^{n} c_j x_j [c^T x]$
  $given \sum_{j=1}^{n} a_{ij} x_j \geq b_i, i = 1, \cdots, m [Ax \geq b]$
  $x_j \geq 0, j = 1, \cdots, n [x \geq 0]$ where a_{ij}, b_i, c_j are given rational numbers.
- Here, A is an $m \times n$ matrix, b is an $m \times 1$ matrix, and x and c are an $n \times 1$ matrices. Note that b is a lower bound on Ax, whereas we cannot indefinitely inflate x since we wish to minimize $c^T x$.

Decision version for testing the upper bound

- The optimization (minimization) problem yields an optimal solution x^*.

Decision version for testing the upper bound

- The optimization (minimization) problem yields an optimal solution x^*.
- If we wish to address the question of membership in P or NP, it helps to formulate decision versions of the *linear programming* problem.

Decision version for testing the upper bound

- The optimization (minimization) problem yields an optimal solution x^*.
- If we wish to address the question of membership in P or NP, it helps to formulate decision versions of the *linear programming* problem.
- Instead of computing x^*, we may ask whether $z^* = c^T x^*$ is at most α, where α is a real number.

Decision version for testing the upper bound

- The optimization (minimization) problem yields an optimal solution x^*.
- If we wish to address the question of membership in P or NP, it helps to formulate decision versions of the *linear programming* problem.
- Instead of computing x^*, we may ask whether $z^* = c^T x^*$ is at most α, where α is a real number.
- Note that we do not know z^* when we are given the decision version instance, denoted by matrices A, b, c and α. Nevertheless, we pose the decision version question "whether $z^* \leq \alpha$".

Decision version for testing the upper bound

- In other words, we may ask whether the optimal value of the objective function is upper bounded by some not too large constant.

Decision version for testing the upper bound

- In other words, we may ask whether the optimal value of the objective function is upper bounded by some not too large constant.
- Supppose we have a 'yes' instance. Then, we are assured that indeed $z^* \leq \alpha$.

Decision version for testing the upper bound

- In other words, we may ask whether the optimal value of the objective function is upper bounded by some not too large constant.
- Supppose we have a 'yes' instance. Then, we are assured that indeed $z^* \leq \alpha$.
- In that case, there must be some $x = a > 0$ that satisfies $Aa \geq b$, and $z^* \leq c^T a = d \leq \alpha$.

Decision version for testing the upper bound

- In other words, we may ask whether the optimal value of the objective function is upper bounded by some not too large constant.
- Supppose we have a 'yes' instance. Then, we are assured that indeed $z^* \leq \alpha$.
- In that case, there must be some $x = a > 0$ that satisfies $Aa \geq b$, and $z^* \leq c^T a = d \leq \alpha$.
- Such an a is a feasible (possibly non-optimal) solution which is a 'witness' that this is a 'yes' instance.

Decision version for testing the upper bound

- In other words, we may ask whether the optimal value of the objective function is upper bounded by some not too large constant.
- Supppose we have a 'yes' instance. Then, we are assured that indeed $z^* \leq \alpha$.
- In that case, there must be some $x = a > 0$ that satisfies $Aa \geq b$, and $z^* \leq c^T a = d \leq \alpha$.
- Such an a is a feasible (possibly non-optimal) solution which is a 'witness' that this is a 'yes' instance.
- The moment we know such a 'witness' a, we set $d = c^T a$, and we can easily check whether $Aa \geq b$ and $c^T a \leq \alpha$, confirming and verifying that z^* is also at most α, that is, $z^* \leq c^T a = d \leq \alpha$.

Decision version is in the class NP

- In other words, we can *verify* efficiently that α is indeed an upper bound on z^*, even though we do not know z^*, simply by checking a 'witness' for the given 'yes' instance.

Decision version is in the class NP

- In other words, we can *verify* efficiently that α is indeed an upper bound on z^*, even though we do not know z^*, simply by checking a 'witness' for the given 'yes' instance.
- This means that the decision problem at hand is indeed in the class NP.

Decision version is in the class NP

- In other words, we can *verify* efficiently that α is indeed an upper bound on z^*, even though we do not know z^*, simply by checking a 'witness' for the given 'yes' instance.
- This means that the decision problem at hand is indeed in the class NP.
- We simply can check efficiently given such a certificate a, that the given instance is indeed a 'yes' instance.

Decision version is in the class NP

- In other words, we can *verify* efficiently that α is indeed an upper bound on z^*, even though we do not know z^*, simply by checking a 'witness' for the given 'yes' instance.
- This means that the decision problem at hand is indeed in the class NP.
- We simply can check efficiently given such a certificate a, that the given instance is indeed a 'yes' instance.
- Is this decision question also in the class co-NP? We will soon answer this question after we define what is known as the *dual* problem of a given (*primal*) linear program.

Linear Programming: Duality

- $$minimize \sum_{j=1}^{n} c_j x_j [c^T x]$$

$$given \sum_{j=1}^{n} a_{ij} x_j \geq b_i, i = 1, \cdots, m [Ax \geq b]$$

$$x_j \geq 0, j = 1, \cdots, n [x \geq 0]$$

Linear Programming: Duality

- $$minimize \sum_{j=1}^{n} c_j x_j [c^T x]$$

  $$given \sum_{j=1}^{n} a_{ij} x_j \geq b_i, i = 1, \cdots, m [Ax \geq b]$$

  $$x_j \geq 0, j = 1, \cdots, n [x \geq 0]$$

- $$maxmize \sum_{i=1}^{m} b_i y_i [b^T y]$$

  $$given \sum_{i=1}^{m} a_{ij} y_i \leq c_j, j = 1, \cdots, n [A^T y \leq c]$$

  $$y_i \geq 0, i = 1, \cdots, m [y \geq 0]$$

Linear Programming: Weak duality

- Here, the lower bounds b_i in the constraints in the primal program define the objective function for maximization in the dual program.

Linear Programming: Weak duality

- Here, the lower bounds b_i in the constraints in the primal program define the objective function for maximization in the dual program.
- Symmetrically, the upper bounds in the constraints of the dual program define the objective function in the primal program.

Linear Programming: Weak duality

- Here, the lower bounds b_i in the constraints in the primal program define the objective function for maximization in the dual program.
- Symmetrically, the upper bounds in the constraints of the dual program define the objective function in the primal program.
- Observe further that the ‘variables’ in $y \geq 0$, in the dual linear program are multipliers of the lower bounds in b of the primal linear program.

Linear Programming: Weak duality

- Here, the lower bounds b_i in the constraints in the primal program define the objective function for maximization in the dual program.
- Symmetrically, the upper bounds in the constraints of the dual program define the objective function in the primal program.
- Observe further that the ‘variables’ in $y \geq 0$, in the dual linear program are multipliers of the lower bounds in b of the primal linear program.
- Even though we maximize the objective function in the dual, which is the dot or inner product of b with the weight- or price- or the variables- vector y, we are well guarded by the upper bounds in $c \geq A^T y$.

Linear Programming: Weak duality

- So, we are ensured that the coefficients of each primal variable x_i, in all the m inequalities of the primal, when weighted by the m multipliers or variables in y of the dual, do not exceed the corresponding cost c_i of the primal.

Linear Programming: Weak duality

- So, we are ensured that the coefficients of each primal variable x_i, in all the m inequalities of the primal, when weighted by the m multipliers or variables in y of the dual, do not exceed the corresponding cost c_i of the primal.
- This ensures that the objective function value in the dual is always below that in the primal, for any pair of feasible solution x and y of the primal and dual, respectively.

Linear Programming: Weak duality

- So, we are ensured that the coefficients of each primal variable x_i, in all the m inequalities of the primal, when weighted by the m multipliers or variables in y of the dual, do not exceed the corresponding cost c_i of the primal.
- This ensures that the objective function value in the dual is always below that in the primal, for any pair of feasible solution x and y of the primal and dual, respectively.
- With this intuition, we now proceed to formally establish the 'weak duality' result below.

Linear Programming: Weak duality

- For feasible solutions x and y to the primal and dual respectively

$$\sum_{j=1}^{n} c_j x_j \geq \sum_{i=1}^{m} b_i y_i [c^T x \geq b^T y]$$

Linear Programming: Weak duality

- For feasible solutions x and y to the primal and dual respectively

$$\sum_{j=1}^{n} c_j x_j \geq \sum_{i=1}^{m} b_i y_i [c^T x \geq b^T y]$$

- $$\sum_{j=1}^{n} c_j x_j \geq \sum_{j=1}^{n} (\sum_{i=1}^{m} a_{ij} y_i) x_j [c^T x \geq x^T A^T y]$$

Linear Programming: Weak duality

- For feasible solutions x and y to the primal and dual respectively

$$\sum_{j=1}^{n} c_j x_j \geq \sum_{i=1}^{m} b_i y_i [c^T x \geq b^T y]$$

- $$\sum_{j=1}^{n} c_j x_j \geq \sum_{j=1}^{n} (\sum_{i=1}^{m} a_{ij} y_i) x_j [c^T x \geq x^T A^T y]$$

- $$\sum_{i=1}^{m} (\sum_{j=1}^{n} a_{ij} x_j) y_i \geq \sum_{i=1}^{m} b_i y_i [y^T A x \geq b^T y]$$

Primal-dual optimality and complementary slackness

- $$x^T A^T y = y^T A x$$

Primal-dual optimality and complementary slackness

- $$x^T A^T y = y^T A x$$

- Feasible solutions x and y for the primal and dual respectively, are both optimal if and only if the following hold

$$\sum_{j=1}^{n} c_j x_j = \sum_{i=1}^{m} b_i y_i [c^T x = b^T y]$$

These are equivalent to the conjunction of the following two conditions of *complementary slackness*.

Primal-dual optimality and complementary slackness

- $$x^T A^T y = y^T A x$$

- Feasible solutions x and y for the primal and dual respectively, are both optimal if and only if the following hold

$$\sum_{j=1}^{n} c_j x_j = \sum_{i=1}^{m} b_i y_i [c^T x = b^T y]$$

These are equivalent to the conjunction of the following two conditions of *complementary slackness*.

- For each $1 \leq j \leq n$ either $x_j = 0$ or $\sum_{i=1}^{m} a_{ij} y_i = c_j$

Primal-dual optimality and complementary slackness

- $$x^T A^T y = y^T A x$$

- Feasible solutions x and y for the primal and dual respectively, are both optimal if and only if the following hold

$$\sum_{j=1}^{n} c_j x_j = \sum_{i=1}^{m} b_i y_i [c^T x = b^T y]$$

These are equivalent to the conjunction of the following two conditions of *complementary slackness*.

- For each $1 \leq j \leq n$ either $x_j = 0$ or $\sum_{i=1}^{m} a_{ij} y_i = c_j$
- For each $1 \leq i \leq m$ either $y_i = 0$ or $\sum_{j=1}^{n} a_{ij} x_j = b_i$

Membership in the class co-NP

- Given a linear programming primal instance A, b, c and α, whether $z^* \geq \alpha$, that is, whether z^* is at least α.

Membership in the class co-NP

- Given a linear programming primal instance A, b, c and α, whether $z^* \geq \alpha$, that is, whether z^* is at least α.
- This question being complementary to the question in the original problem, establishing the membership in NP for this question would place the original problem in the class co-NP.

Membership in the class co-NP

- Given a linear programming primal instance A, b, c and α, whether $z^* \geq \alpha$, that is, whether z^* is at least α.
- This question being complementary to the question in the original problem, establishing the membership in NP for this question would place the original problem in the class co-NP.
- This is easy to show using a similar argument applied to suitable feasible solutions of the dual linear program that have lower bounded objective function values.

Membership in the class co-NP

- Given a linear programming primal instance A, b, c and α, whether $z^* \geq \alpha$, that is, whether z^* is at least α.
- This question being complementary to the question in the original problem, establishing the membership in NP for this question would place the original problem in the class co-NP.
- This is easy to show using a similar argument applied to suitable feasible solutions of the dual linear program that have lower bounded objective function values.
- Using such solutions of the dual as ‘certificates’ or ‘witnesses’, ‘yes’ instances of this new problem can be shown to be checkable in polynomial time.

Dual fitting analysis technique for the greedy set cover

- The problem of minimum set cover is as follows.
  $minimize \sum_{S\in\mathcal{S}} c(S)x_S$ subject to
  $\sum_{S:e\in S} x_S \geq 1, e \in U$
  $x_S \in \{0,1\}, S \in \mathcal{S}$.

Dual fitting analysis technique for the greedy set cover

- The problem of minimum set cover is as follows.
$minimize \sum_{S \in \mathcal{S}} c(S) x_S$ subject to
$\sum_{S : e \in S} x_S \geq 1, e \in U$
$x_S \in \{0, 1\}, S \in \mathcal{S}$.
- This is a 0-1 integer program.

Dual fitting analysis technique for the greedy set cover

- The problem of minimum set cover is as follows.
  $minimize \sum_{S \in \mathcal{S}} c(S) x_S$ subject to
  $\sum_{S: e \in S} x_S \geq 1, e \in U$
  $x_S \in \{0, 1\}, S \in \mathcal{S}$.
- This is a 0-1 integer program.
- The *LP-relaxation* of this integer program is the following *primal* linear program.
  $minimize \sum_{S \in \mathcal{S}} c(S) x_S$ subject to
  $\sum_{S: e \in S} x_S \geq 1, e \in U$
  $x_S \geq 0, S \in \mathcal{S}$.

Dual fitting analysis technique for the greedy set cover

- The problem of minimum set cover is as follows.
  $minimize \sum_{S \in \mathcal{S}} c(S)x_S$ subject to
  $\sum_{S:e \in S} x_S \geq 1, e \in U$
  $x_S \in \{0, 1\}, S \in \mathcal{S}$.
- This is a 0-1 integer program.
- The *LP-relaxation* of this integer program is the following *primal* linear program.
  $minimize \sum_{S \in \mathcal{S}} c(S)x_S$ subject to
  $\sum_{S:e \in S} x_S \geq 1, e \in U$
  $x_S \geq 0, S \in \mathcal{S}$.
- The *dual* linear program is
  $maximize \sum_{e \in U} y_e$ subject to
  $\sum_{e:e \in S} y_e \leq c(S), S \in \mathcal{S}, y_e \geq 0, e \in U$

The dual lower bound

- We know that the optimal cost OPT of the set cover is at least the optimal cost OPT_f of the primal linear program in the LP relaxation.

The dual lower bound

- We know that the optimal cost OPT of the set cover is at least the optimal cost OPT_f of the primal linear program in the LP relaxation.
- We also know that the cost of any feasible solution to the dual linear program is no more than OPT_f, which in turn is no more than OPT.

The dual lower bound

- We know that the optimal cost OPT of the set cover is at least the optimal cost OPT_f of the primal linear program in the LP relaxation.
- We also know that the cost of any feasible solution to the dual linear program is no more than OPT_f, which in turn is no more than OPT.
- The optimal costs of the primal and dual linear programs are both OPT_f.

The dual lower bound

- We know that the optimal cost OPT of the set cover is at least the optimal cost OPT_f of the primal linear program in the LP relaxation.
- We also know that the cost of any feasible solution to the dual linear program is no more than OPT_f, which in turn is no more than OPT.
- The optimal costs of the primal and dual linear programs are both OPT_f.
- When we choose the next element $e_i \in S = \{e_1, e_2, \cdots, e_k\}$ of the k elements of a set S in the greedy set cover heuristic, the $price(e_i)$ is no more than $\frac{c(S)}{k-i+1}$, as we now demonstrate.

The greedy set cover prices

- The main argument is that the element e_i may be incorporated due to the inclusion of either the set S itself or some other set.

The greedy set cover prices

- The main argument is that the element e_i may be incorporated due to the inclusion of either the set S itself or some other set.
- If S is itself chosen then there are $k-i+1$ new elements $e_i, ..., e_k$ to be included with cost effectivity $\frac{c(S)}{k-i+1}$, the assigned value of $price(e_j)$, $i \leq j \leq k$.

The greedy set cover prices

- The main argument is that the element e_i may be incorporated due to the inclusion of either the set S itself or some other set.
- If S is itself chosen then there are $k - i + 1$ new elements $e_i, ..., e_k$ to be included with cost effectivity $\frac{c(S)}{k-i+1}$, the assigned value of $price(e_j)$, $i \leq j \leq k$.
- Clearly, $price(e_j) = \frac{c(S)}{k-i+1} \leq \frac{c(S)}{k-j+1}$, $i \leq j \leq k$.

The greedy set cover prices

- The main argument is that the element e_i may be incorporated due to the inclusion of either the set S itself or some other set.
- If S is itself chosen then there are $k - i + 1$ new elements $e_i, ..., e_k$ to be included with cost effectivity $\frac{c(S)}{k-i+1}$, the assigned value of $price(e_j)$, $i \leq j \leq k$.
- Clearly, $price(e_j) = \frac{c(S)}{k-i+1} \leq \frac{c(S)}{k-j+1}$, $i \leq j \leq k$.
- Otherwise, some other set includes e_i with lower cost effectivity, such that $price(e_i) \leq \frac{c(S)}{k-i+1}$, as per the greedy algorithm.

The greedy set cover prices

- Now setting the variable y_e of the dual linear program for each $e \in U$ to $\frac{price(e)}{H(n)}$, we observe that

$$y_{e_i} \leq \frac{1}{H(n)} \cdot \frac{c(S)}{k - i + 1}$$

for each of the k elements $e_i \in S$.

The greedy set cover prices

- Now setting the variable y_e of the dual linear program for each $e \in U$ to $\frac{price(e)}{H(n)}$, we observe that

$$y_{e_i} \leq \frac{1}{H(n)} \cdot \frac{c(S)}{k-i+1}$$

  for each of the k elements $e_i \in S$.

- So,

$$\sum_{i=1}^{k} y_{e_i} \leq \frac{c(S)}{H(n)}.(\frac{1}{k} + \frac{1}{k-1} + \cdots + \frac{1}{1}) = \frac{H(k)}{H(n)}.c(S) \leq c(S)$$

The greedy set cover prices

- Now setting the variable y_e of the dual linear program for each $e \in U$ to $\frac{price(e)}{H(n)}$, we observe that

$$y_{e_i} \leq \frac{1}{H(n)} \cdot \frac{c(S)}{k-i+1}$$

for each of the k elements $e_i \in S$.

- So,

$$\sum_{i=1}^{k} y_{e_i} \leq \frac{c(S)}{H(n)}.(\frac{1}{k} + \frac{1}{k-1} + \cdots + \frac{1}{1}) = \frac{H(k)}{H(n)}.c(S) \leq c(S)$$

- So, the constraints in the dual linear program are satisfied establishing the feasibility of the solution with y_e values as assigned above. Now we further observe that

$$\sum_{e \in U} price(e) = H(n)(\sum_{e \in U} y_e) \leq H(n).OPT_f \leq H(n).OPT$$

Linear Programming and Weighted Vertex Cover

- We develop approximation algorithms based on *linear programming*.

Linear Programming and Weighted Vertex Cover

- We develop approximation algorithms based on *linear programming*.
- Consider the weighted version of the *vertex cover* problem on weighted undirected graphs.

Linear Programming and Weighted Vertex Cover

- We develop approximation algorithms based on *linear programming*.
- Consider the weighted version of the *vertex cover* problem on weighted undirected graphs.
- A *linear program* has a system $Ax \geq b$ of inequalities called *constraints*, and an *objective* function $c^T x$.

Linear Programming and Weighted Vertex Cover

- We develop approximation algorithms based on *linear programming*.
- Consider the weighted version of the *vertex cover* problem on weighted undirected graphs.
- A *linear program* has a system $Ax \geq b$ of inequalities called *constraints*, and an *objective* function $c^T x$.
- We need to minimize $c^T x$ over all *positive vectors* $x \geq 0$, satisfying the set of constraints.

Linear Programming and Weighted Vertex Cover

- We develop approximation algorithms based on *linear programming*.
- Consider the weighted version of the *vertex cover* problem on weighted undirected graphs.
- A *linear program* has a system $Ax \geq b$ of inequalities called *constraints*, and an *objective* function $c^T x$.
- We need to minimize $c^T x$ over all *positive vectors* $x \geq 0$, satisfying the set of constraints.
- The set of constraints represents the intersection of half-spaces, which is a convex region of multi-dimensional space, called the *feasible* region.

Linear Programming and Weighted Vertex Cover

- We develop approximation algorithms based on *linear programming*.
- Consider the weighted version of the *vertex cover* problem on weighted undirected graphs.
- A *linear program* has a system $Ax \geq b$ of inequalities called *constraints*, and an *objective* function $c^T x$.
- We need to minimize $c^T x$ over all *positive vectors* $x \geq 0$, satisfying the set of constraints.
- The set of constraints represents the intersection of half-spaces, which is a convex region of multi-dimensional space, called the *feasible* region.
- Optima of linear objective functions like $c^T x$ can occur only at vertices of this convex feasible region.

Formulation with weights for vertices

- Being more precise, the problem we define is as follows. *Given an* $m \times n$ *matrix* A*, and vectors* $b \in \mathcal{R}^m$ *and* $c \in \mathcal{R}^n$*, find a vector* $x \in \mathcal{R}^n$ *solving the optimization problem min*$\{c^T x$ *such that* $x \geq 0$ *and* $Ax \geq b\}$.

Formulation with weights for vertices

- Being more precise, the problem we define is as follows. *Given an* $m \times n$ *matrix* A, *and vectors* $b \in \mathcal{R}^m$ *and* $c \in \mathcal{R}^n$, *find a vector* $x \in \mathcal{R}^n$ *solving the optimization problem min*$\{c^T x$ *such that* $x \geq 0$ *and* $Ax \geq b\}$.
- Each vertex i has a positive weight w_i. We say that the weight of a set of vertices is the sum of weights of its vertices.

Formulation with weights for vertices

- Being more precise, the problem we define is as follows. *Given an* $m \times n$ *matrix* A, *and vectors* $b \in \mathcal{R}^m$ *and* $c \in \mathcal{R}^n$, *find a vector* $x \in \mathcal{R}^n$ *solving the optimization problem min*$\{c^T x$ *such that* $x \geq 0$ *and* $Ax \geq b\}$.
- Each vertex i has a positive weight w_i. We say that the weight of a set of vertices is the sum of weights of its vertices.
- We wish to compute a vertex cover with at most twice the optimal weight in polynomial time.

Use of indicator variables for vertex cover

- We use an indicator or *decision* variable x_i for inclusion of the ith vertex in the vertex cover.

Use of indicator variables for vertex cover

- We use an indicator or *decision* variable x_i for inclusion of the ith vertex in the vertex cover.
- The minimum weighted vertex cover will minimize

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

such that

$$x_i + x_j \geq 1, (i, j) \in E$$

and

$$x_i \in \{0, 1\}, i \in V$$

Discrete Integer Linear Program

- We can rewrite the problem formally as

$$Ax \geq 1$$

$$1 \geq x \geq 0$$

where the integer 0-1 matrix A has one row for each edge and one column for each vertex and $A[e, i] = 1$ whenever vertex i is in edge e and 0, otherwise.

Discrete Integer Linear Program

- We can rewrite the problem formally as

$$Ax \geq 1$$

$$1 \geq x \geq 0$$

where the integer 0-1 matrix A has one row for each edge and one column for each vertex and $A[e, i] = 1$ whenever vertex i is in edge e and 0, otherwise.

- We need to solve the optimization problem $\min\{w^T x$ such that $1 \geq x \geq 0$ and $Ax \geq 1, x \in \{0, 1\}\}$.

Discrete Integer Linear Program

- We can rewrite the problem formally as

$$Ax \geq 1$$

$$1 \geq x \geq 0$$

where the integer 0-1 matrix A has one row for each edge and one column for each vertex and $A[e, i] = 1$ whenever vertex i is in edge e and 0, otherwise.

- We need to solve the optimization problem $\min\{w^T x$ such that $1 \geq x \geq 0$ and $Ax \geq 1, x \in \{0, 1\}\}$.

- We have reduced the optimization version of the minimum weighted vertex cover problem to the linear programming problem where we require the solutions (for x_i) to be from $\{0, 1\}$.

NP-hardness of ILP and Weighted Vertex Cover

- This problem is called *0-1 integer programming*, and due to the NP-hardness of the vertex cover problem, this problem is also NP-hard.

NP-hardness of ILP and Weighted Vertex Cover

- This problem is called *0-1 integer programming*, and due to the NP-hardness of the vertex cover problem, this problem is also NP-hard.
- Why is the *decision version* of this 0-1 integer programming problem also in the class NP?

Linear Programming relaxation for the ILP

- Let w_{LP} be the optimal weight for this optimization problem. For the corresponding solution x^*, the components x_i^* may be non-integral.

Linear Programming relaxation for the ILP

- Let w_{LP} be the optimal weight for this optimization problem. For the corresponding solution x^*, the components x_i^* may be non-integral.
- One way to get an integer solution is to round the fractional solutions which are in the range between 0 and 1.

Linear Programming relaxation for the ILP

- Let w_{LP} be the optimal weight for this optimization problem. For the corresponding solution x^*, the components x_i^* may be non-integral.
- One way to get an integer solution is to round the fractional solutions which are in the range between 0 and 1.
- If $x_i^* \geq \frac{1}{2}$, only then we include i in S. Why do we get a vertex cover?

Linear Programming relaxation for the ILP

- Let w_{LP} be the optimal weight for this optimization problem. For the corresponding solution x^*, the components x_i^* may be non-integral.
- One way to get an integer solution is to round the fractional solutions which are in the range between 0 and 1.
- If $x_i^* \geq \frac{1}{2}$, only then we include i in S. Why do we get a vertex cover?
- This way we get an approximate vertex cover, whose total weight will now be shown to be at most twice the optimal.

The lower bound and ensuing approximation cap

- First observe that $w_{LP} \leq w(S^*)$, where S^* is any optimal weighted vertex cover.

The lower bound and ensuing approximation cap

- First observe that $w_{LP} \leq w(S^*)$, where S^* is any optimal weighted vertex cover.
- This is because the optimal vertex cover is a special case where the solutions are integral and relaxing this restriction cannot worsen the solution with respect to optimization.

The lower bound and ensuing approximation cap

- First observe that $w_{LP} \leq w(S^*)$, where S^* is any optimal weighted vertex cover.
- This is because the optimal vertex cover is a special case where the solutions are integral and relaxing this restriction cannot worsen the solution with respect to optimization.
- Also, $w(S^*) \geq w_{LP} = w^T x^* = \sum_i w_i x_i^* \geq \sum_{i \in S} w_i x_i^* \geq \frac{1}{2} \sum_{i \in S} w_i = \frac{1}{2} w(S)$.

The lower bound and ensuing approximation cap

- First observe that $w_{LP} \leq w(S^*)$, where S^* is any optimal weighted vertex cover.
- This is because the optimal vertex cover is a special case where the solutions are integral and relaxing this restriction cannot worsen the solution with respect to optimization.
- Also, $w(S^*) \geq w_{LP} = w^T x^* = \sum_i w_i x_i^* \geq \sum_{i \in S} w_i x_i^* \geq \frac{1}{2} \sum_{i \in S} w_i = \frac{1}{2} w(S)$.
- So, we have $w(S) \leq 2w_{LP} \leq 2w(S^*)$.