

Approximation Algorithms

Optimization Problem

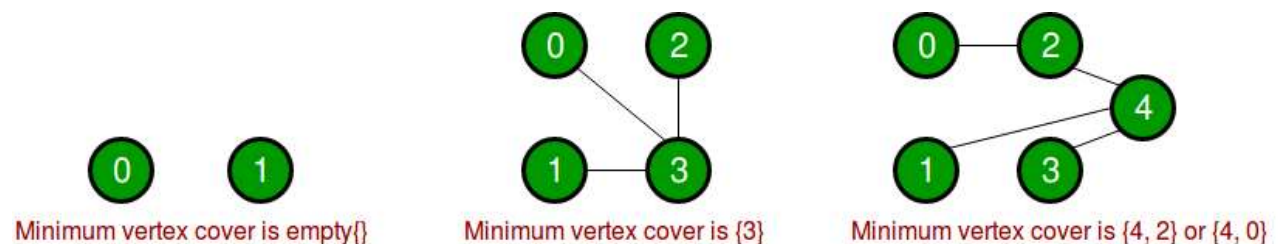
- An optimization problem is the problem of finding the best solution from all feasible solutions.
- The objective may be either minimum or maximum depending on the problem considered.
- A large number of optimization problems which are required to be solved in practice are NP-hard.
- For such problems, it is not possible to design algorithms that can find exactly optimal solution to all instances of the problem in polynomial time.

Approximation Algorithms

- Algorithm which returns solutions that are guaranteed to be close to an optimal solution.
- Guaranteed to run in polynomial time.
- **Approximation algorithms** are typically used when finding an optimal solution is intractable, but can also be used in some situations where a near-optimal solution can be found quickly and an exact solution is not **needed**.
- **Approximation algorithms** are efficient algorithms that find approximate solutions to optimization problems (in particular NP-hard problems).
- This technique does not guarantee the best solution. Guarantees near-optimal solution. The goal of an approximation algorithm is to come as close as possible to the optimum value in a reasonable amount of time which is at the most polynomial time.

1. Vertex cover problem

- A vertex cover of an undirected graph is a subset of its vertices such that for every edge (u, v) of the graph, either 'u' or 'v' is in vertex cover.
- Although the name is Vertex Cover, the set covers all edges of the given graph.
- **Given an undirected graph, the vertex cover problem is to find minimum size vertex cover.**
- Vertex cover problem is a known NP Complete problem, i.e., there is no polynomial time solution for this. There are approximate polynomial time algorithms to solve the problem though.
- Examples:

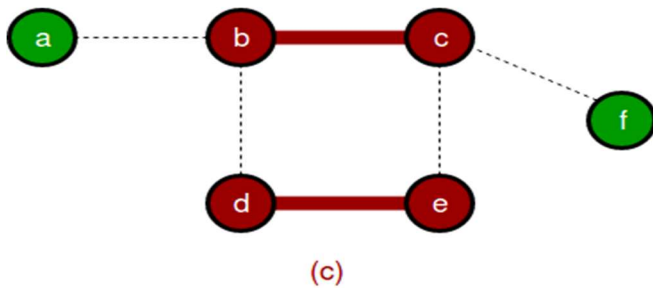
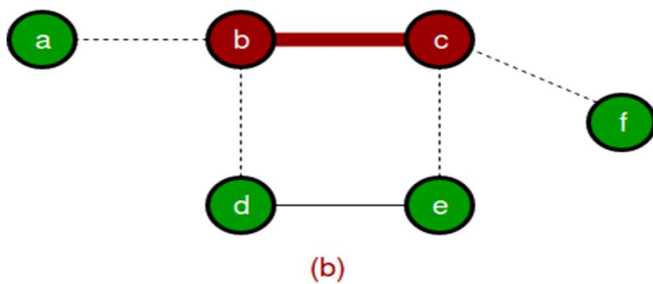
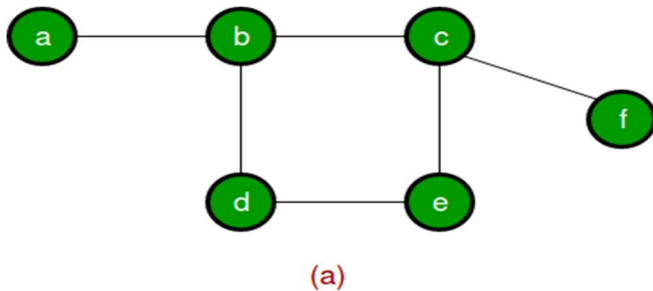


Approximate Greedy Algorithm for Vertex Cover

- 1) Initialize the result as {}
- 2) consider a set of all edges in given graph. Let the set be E.
- 3) Do following while E is not empty
 - ...a) Pick an arbitrary edge (u, v) from set E and add 'u' and 'v' to result
 - ...b) Remove all edges from E which are either incident on u or v.
- 4) Return result

- It can be proved that the above approximate algorithm never finds a vertex cover whose size is more than twice the size of minimum possible vertex cover (CLOSE TO AN OPTIMAL SOLUTION).
- Time Complexity of above algorithm is $O(V + E)$.

Example – 1:



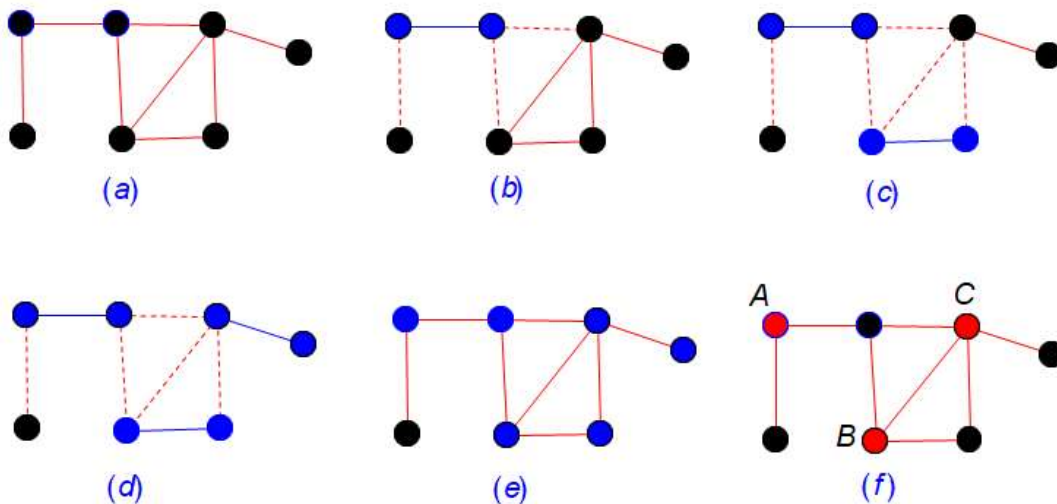
Minimum Vertex Cover is $\{b, c, d\}$ or $\{b, c, e\}$

Size of minimum vertex cover = 3.

Approximate Vertex cover = $\{b, c, d, e\}$ whose size = 4.

This approximate solution is near-optimal solution (or) close to the optimal solution as its size is not more than twice the size of the minimum vertex cover.

Example – 2:



Approximate Vertex cover size = 6 (**blue colored** vertices)

Minimum vertex cover = 3 (**red colored** vertices)

2. Set cover problem

- Given a universe U of n elements, a collection of subsets of U say $S = \{S_1, S_2, \dots, S_m\}$ where every subset S_i has an associated cost. Find a minimum cost sub collection of S that covers all elements of U .
- Example:

$U = \{1, 2, 3, 4, 5\}$

$S = \{S_1, S_2, S_3\}$

$S_1 = \{4, 1, 3\}$, $\text{Cost}(S_1) = 5$

$S_2 = \{2, 5\}$, $\text{Cost}(S_2) = 10$

$S_3 = \{1, 4, 3, 2\}$, $\text{Cost}(S_3) = 3$

Minimum cost of set cover is 13 and set cover is $\{S_2, S_3\}$

There are two possible set covers $\{S_1, S_2\}$ with cost 15 and $\{S_2, S_3\}$ with cost 13.

- Set Cover is NP-Hard: There is no polynomial time solution available for this problem. There is a polynomial time Greedy approximate algorithm.

Approximate Greedy Algorithm:

Let U be the universe of elements, $\{S_1, S_2, \dots, S_m\}$ be collection of subsets of U and $\text{Cost}(S_1), \text{Cost}(S_2), \dots, \text{Cost}(S_m)$ be costs of subsets.

- 1) Let I represents set of elements included so far. Initialize $I = \{\}$
- 2) Do following while I is not same as U .
 - a) Find the set S_i in $\{S_1, S_2, \dots, S_m\}$ whose cost effectiveness is smallest, i.e., the ratio of cost $C(S_i)$ and number of newly added elements is minimum.

Basically we pick the set for which following value is minimum.
$$\text{Cost}(S_i) / |S_i - I|$$
 - b) Add elements of above picked S_i to I , i.e., $I = I \cup S_i$

Example:

$$U = \{1, 2, 3, 4, 5\}$$

$$S = \{S_1, S_2, S_3\}$$

$$S_1 = \{4, 1, 3\}, \quad \text{Cost}(S_1) = 5$$

$$S_2 = \{2, 5\}, \quad \text{Cost}(S_2) = 10$$

$$S_3 = \{1, 4, 3, 2\}, \quad \text{Cost}(S_3) = 3$$

First Iteration:

$$I = \{\}$$

The per new element cost for $S_1 = \text{Cost}(S_1)/|S_1 - I| = 5/3$

The per new element cost for $S_2 = \text{Cost}(S_2)/|S_2 - I| = 10/2$

The per new element cost for $S_3 = \text{Cost}(S_3)/|S_3 - I| = 3/4$

Since S_3 has minimum value S_3 is added, I becomes $\{1,4,3,2\}$.

Set cover = $\{S_3\}$

Set cover cost = 3

Second Iteration:

$I = \{1,4,3,2\}$

The per new element cost for $S_1 = \text{Cost}(S_1)/|S_1 - I| = 5/0$

Note that S_1 doesn't add any new element to I .

The per new element cost for $S_2 = \text{Cost}(S_2)/|S_2 - I| = 10/1$

Note that S_2 adds only 5 to I .

S_2 is added, $I = \{1, 4, 3, 2, 5\}$

Set cover = $\{S_2, S_3\}$

Set cover cost = 13

Algorithm terminates as I is same as U .

In this example, the approximate greedy algorithm provides the optimal solution:

Set cover = $\{S_2, S_3\}$

Set cover cost = 13

But it may not provide optimal solution all the time.

Example:

$$U = \{1, 2, 3, \dots, 13\}$$

$$S = \{S_1, S_2, S_3, S_4, S_5\}$$

$$S_1 = \{1, 2\}$$

$$S_2 = \{2, 3, 4, 5\}$$

$$S_3 = \{6, 7, 8, 9, 10, 11, 12, 13\}$$

$$S_4 = \{1, 3, 5, 7, 9, 11, 13\}$$

$$S_5 = \{2, 4, 6, 8, 10, 12, 13\}$$

Let the cost of every set be 10.

First Iteration:

$$I = \{\}$$

$$\text{The per new element cost for } S_1 = \text{Cost}(S_1) / |S_1 - I| = 10/2$$

$$\text{The per new element cost for } S_2 = \text{Cost}(S_2) / |S_2 - I| = 10/4$$

$$\text{The per new element cost for } S_3 = \text{Cost}(S_3) / |S_3 - I| = 10/8$$

$$\text{The per new element cost for } S_4 = \text{Cost}(S_4) / |S_4 - I| = 10/7$$

$$\text{The per new element cost for } S_5 = \text{Cost}(S_5) / |S_5 - I| = 10/7$$

Since S_3 has minimum value S_3 is added, I becomes $\{6, 7, 8, 9, 10, 11, 12, 13\}$.

$$\text{Set cover} = \{S_3\}$$

$$\text{Set cover cost} = 10$$

Second Iteration:

$$I = \{6, 7, 8, 9, 10, 11, 12, 13\}$$

The per new element cost for $S_1 = \text{Cost}(S_1)/|S_1 - I| = 10/2$

Note that S_1 adds 2 new elements to I .

The per new element cost for $S_2 = \text{Cost}(S_2)/|S_2 - I| = 10/4$

Note that S_2 adds 4 new elements to I .

The per new element cost for $S_4 = \text{Cost}(S_4)/|S_4 - I| = 10/3$

Note that S_4 adds 3 new elements to I .

The per new element cost for $S_5 = \text{Cost}(S_5)/|S_5 - I| = 10/2$

Note that S_5 adds 2 new elements to I .

Since S_2 has minimum value S_2 is added, I becomes $\{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13\}$.

Set cover = $\{S_3, S_2\}$

Set cover cost = 20

Third Iteration:

$I = \{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13\}$

The per new element cost for $S_1 = \text{Cost}(S_1)/|S_1 - I| = 10/1$

Note that S_1 adds 1 new element to I .

The per new element cost for $S_4 = \text{Cost}(S_4)/|S_4 - I| = 10/1$

Note that S_4 adds 1 new element to I .

The per new element cost for $S_5 = \text{Cost}(S_5)/|S_5 - I| = 10/0$

Note that S_5 adds 0 new elements to I .

Since S_1 and S_4 has same minimum value S_1 or S_4 is added, I becomes $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13\}$.

Set cover = $\{S_3, S_2, S_1\}$ or $\{S_3, S_2, S_4\}$

Set cover cost = 30

Algorithm terminates as I is same as U .

In this example, the approximate greedy algorithm provides the near-optimal solution:

Set cover = $\{S_3, S_2, S_1\}$ or $\{S_3, S_2, S_4\}$

Set cover cost = 30

The optimal solution is $\{S_4, S_5\}$ whose cost is 20.