Set Cover Problem

Prepared By:

Pallavi Zate
21MPC1014
SCOPE

Updated By:

Dr. Shruti Mishra
SCOPE

Contents

Definition

- "You must select a minimum number [of any size set] of these sets so that the sets you have picked contain all the elements that are contained in any of the sets in the input." Additionally, you want to minimize the cost of the sets.
- Given a universe U of n elements, a collection of subsets of U say S = {S1, S2…,Sn} where every subset Si has an associated cost. Find a minimum cost sub-collection of S that covers all elements of U.
- The decision version of set covering is NP-complete, and the optimization/search version of set cover is NP-hard.

Example

$U = \{1,2,3,4,5\}$, $S = \{S_1,S_2,S_3\}$

$S1 = \{4,1,3\}$, $Cost(S_1) = 5$

$S2 = \{2,5\}$, $Cost(S_2) = 10$

$S3 = \{1,4,3,2\}$, $Cost(S_3) = 3$

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

and $\{S_2, S_3\}$ with cost 13.

So, here Minimum cost of set cover is 13 and set cover is {S2, S3}.

Why it is Useful?

- Edge Covering
- Vertex Covering

Greedy Algorithm

Let U be the universe of elements, {S1, S2, … Sm } be collection of subsets of U and Cost(S1), C(S2), … Cost(Sm) 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 Si in {S1, S2, ... Sm } whose cost effectiveness is smallest, i.e., the ratio of cost C(Si) and number of newly added elements is minimum.

Basically we pick the set for which following value is minimum.

Cost(Si) / |Si - I|

b) Add elements of above picked Si to I, i.e., I = I U Si

Example

Let us consider the above example to understand Greedy Algorithm.

- First Iteration:

I = {}

The per new element cost for S1 = Cost(S1)/|S1 – I| = 5/3

The per new element cost for S2 = Cost(S2)/|S2 – I| = 10/2

The per new element cost for S3 = Cost(S3)/|S3 – I| = ¾

Since S3 has minimum value S3 is added, I becomes {1,4,3,2}.

Contd..

- Second Iteration:

I = {1,4,3,2}

The per new element cost for S1 = Cost(S1)/|S1 – I| = 5/0 that S1 doesn't add any new element to I.

The per new element cost for S2 = Cost(S2)/|S2 – I| = 10/1

Note that S2 adds only 5 to I.

The greedy algorithm provides the optimal solution for above example, but it may not provide optimal solution all the time. Consider the following example.

Contd..

- S1 = {1, 2}
- S2 = {2, 3, 4, 5}
- S3 = {6, 7, 8, 9, 10, 11, 12, 13}
- S4 = {1, 3, 5, 7, 9, 11, 13}
- S5 = {2, 4, 6, 8, 10, 12, 13}

- Let the cost of every set be same.

- The greedy algorithm produces result as {S3, S2, S1}

- The optimal solution is {S4, S5}

Proof

Proof that the above greedy algorithm is log(n)approximate.

- Let OPT be the cost of optimal solution.
- Say (k-1) elements are covered before an iteration of above greedy algorithm. *T* be cost of the k^{th} element <= OPT / (n-k+1). (Note that cost of an element is evaluated by cost of its set divided by number of elements added by its set).
- How did we get this result?
- Since, k^{th} element is not covered yet, there is a S_i that has not been covered before the current step of greedy algorithm and it is there in OPT. Since, greedy algorithm picks the most cost effective S_i, per-element-cost in the picked set must be smaller than OPT divided by remaining elements. Therefore cost of k^{th} element <= OPT/|U-I| (Note that U-I is set of not yet covered elements in Greedy Algorithm).
- The value of |U-I| is n - (k-1) which is n-k+1.

Cost of Greedy Algorithm

Cost of Greedy Algorithm = Sum of costs of n elements

[putting k = 1, 2..n in above formula]

$<= (OPT/n + OPT/(n-1) + ... + OPT/n)$

$<= OPT(1 + 1/2 + ... 1/n)$

[Since $1 + 1/2 + .. 1/n = Log\ n$]

$<= OPT * logn$