Approximation Algorithm

Prof. Palash Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Week – 01

Lecture 06

Lecture 06 : Greedy algorithm for Weighted Set Cover

So, till now we have seen many algorithms for approximating the weighted vertex weighted set cover and all of them has an approximation factor of f. The techniques are linear programming rounding deterministic and then we have seen dual rounding and in the last class we have seen a combinatorial algorithm which is based on the primal dual method. So, today we will see another algorithm with incomparable approximation ratio for set cover and that is greedy a greedy algorithm. So, let us see this algorithm is quite popular algorithm for set cover greedy algorithm. So, what is greedy as you know greedy algorithm is a typical in iterative algorithm where in every step we pick something in the solution which looks the best at the current situation and that is how we iteratively build towards the solution. So, we will do the same thing here.

So, let me write the pseudocode first as typical greedy algorithm this here in this algorithm also we start with an empty solution and iteratively pick or build the solution in a greedy way we pick something in the solution which looks best in the current situation. So, we have the sets recall what is we have input is and universe U which is $\{e_1,\dots,e_n\}$ and collection of sets $S_1,\dots,S_m$ with weights $w_1,\dots,w_m$. So, what is the greedy choice here? So, which set sounds best? So, in the first iteration say.

So, we have m sets each with particular weights. So, it makes sense to pick pick the set which covers the maximum number of elements and also it has lowest cost. It has two things know we need to minimize cost or minimize total weight of the solution, but at the same time we also want to maximize the we want to cover everything in the solution everything in the universe. So, it one natural greedy choice is to pick the set which covers the elements with least average cost. So, for that while I is not a set cover what do we do? For we pick the set which covers thus elements with minimum average weight or average cost.

So, l is suppose that set which is argmin, is the argument which minimizes the expression. In this case I will write an expression in terms of j and argument is that particular index j which minimizes that expression. What is that expression? I want for each set the jth set can cover the elements in S_j at a cost total cost of w_j. So, the set S_j can cover $|S_j|$ elements at a average cost of $\frac{w_j}{|S_j|}$ and I will pick the set which covers the elements with least average cost.

Now, you see after first iteration some element got covered in the next iteration which set I should pick the again a greedy choice is that set which covers the uncovered elements, some elements are already covered in by the first set that I have picked. So, in the next iteration it makes sense to pick that set which covers the uncovered elements with least average cost. So, we need to we need to sort of in some sense get rid of the elements which are which have got covered and focus only on uncovered elements, because we want to cover the currently uncovered elements with least average cost. So, for that we maintain a set $\hat{S}_j$ which is initialized to S_j for all $j \in [m]$. And in $\hat{S}_j$ we are storing we are removing the elements that are got covered. So, this is $\hat{S}_j$ minus in this iteration the elements that got covered is $\hat{S}_l$. So, $\hat{S}_l$ these elements I am removing for all $j \in [m]$. ok and here also I will only focus on elements that are newly uncovered elements that are getting covered. So, this is the thing.

So, in $\hat{S}_j$ in every iteration it stores only the uncovered elements and in every iteration I am picking that set which covers the maximum number of uncovered elements maximum number of uncovered elements with least average cost. And, then we return the set cover what is the set cover I is the indices. So, this is S_i such that $i \in I$. So, this is the set cover this I output. What we will show is that this is a $\log n$ factor approximation algorithm.

So, theorem above algorithm which clearly runs in polynomial time because in every iteration at least one element is getting covered or at least one set is getting picked and so, it can iterate over at most minimum of m or n iteration minimum of m and n iterations. The above algorithm has an approximation factor of H_n where H_n is the n-th harmonic number. $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}$ ok. To prove this we need a fact which is easy to prove you can take it as a homework to prove it. Given positive numbers real numbers $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$.

we have $\frac{a_1 + a_2 + \ldots + a_n}{b_1 + b + 2 + \ldots + b_n}$ this is less than equal to $max_{i \in [n]} \frac{a_i}{b_i}$ and this is greater than equal to $min_{i \in [n]} \frac{a_i}{b_i}$ ok. So, before proving this let me tell you the high level idea let me give a high level idea. The high level idea is as follows, if we know that the optimal solution can cover all elements at an average cost of total cost is of. and the number of elements is n. So, the average cost is opt by n.

Hence, there must exist at the first iteration there must exist at least one set which can cover the elements in it with average cost less than equal to n. Hence, in the beginning of the algorithm there exists a set which covers its elements with average cost at most opt by n. ok and this argument we repeat. So, suppose k elements get covered in the next iteration in the first iteration. So, in the next iteration.

So, suppose k elements gets covered in the first iteration ok. Then again we observe that the optimal solution covers this remaining n minus k elements at an average cost of opt by at most $\frac{opt}{n-k}$. Then the optimal solution covers the remaining n - k elements at an average cost of at most $\frac{opt}{n-k}$. And, hence the set that our algorithm is picking in the second iteration that also has an average cost of at most n - k and so on. So, and hence the set that the algorithm peaks in the second iteration has an average cost considering only elements which are not covered in the first iteration. Considering only those elements that are not covered in the first iteration. cost at most $\frac{opt}{n-k}$ and so on. So, what is the ALG? So, in the first iteration it has covered k elements. So, this alg is less than equal to k if the first iteration if it say covers say k_1 element then this $k_1\frac{opt}{n}$ this is the total cost of the elements covered in the first iteration then $k_2\frac{opt}{n-k_1}$ and so on and this can be shown to be less than equal to $H_n \times opt$ this is the high level idea. So, with this high level idea let us now formalize the proof.

Now, formal proof of the theorem. So, let n_k be the number of uncovered elements in the beginning of kth iteration ok. And suppose the algorithm terminates after l iterations. Then what we have is n_1 which is the number of uncovered elements in the beginning of the first iteration this is equal to n and n_l plus which is the number of uncovered elements the beginning of $l+1$ is iteration, but the algorithm terminates after l iteration. So, this must be 0 and so, now, let k be any arbitrary iterations ok and .

So, suppose the algorithm picks the set S_k in the kth iteration. let us rename the sets. So, that we can assume that S_1 be the set picked in the first iteration, S_2 be the set picked in the second iteration and so on. So, what do we know? We know that opt then we have Now, focus at the beginning of kth iteration the beginning of kth iteration n_k

is the number of uncovered elements and opt can cover them the that those collection of sets can cover them as an average cost of $\frac{opt}{n_k}$ ok, but we have picked S_j .

So, S_j that set can cover them at an average cost of w_k by the number of uncovered elements that the set S_k is covering in the beginning of kth iteration. What is that? That number is nothing, but this is $n_k - n_{k+1}$. This is exactly the number of elements that the set S_k was covering in the beginning of kth iteration. So, this because of the greedy choice this must be less than equal to this.

So, what do you have that W k is less than equal to $\frac{n_k - n_{k+1}}{n_k} opt$ ok. Now, what is ALG? we have renamed the sets. So, that we have assumed without loss of generality that the algorithm picks set S_1 in the first iteration set S_2 in the second iteration and so on. And we have l iterations this is $w_1 + w_2 + \ldots + w_l$. So, this is the weight of the sum of the weights of the sets that the algorithm outputs.

this is $\sum_{k=1}^{l} \frac{n_k - n_{k+1}}{n_k} opt$, opt goes out. This is less than equal to opt $\sum_{k=1}^{l}$ first term let it keep it n_k second term instead of n_k I write $\frac{1}{n_{k-1}}$.

So, I am reducing the denominator. So, the number increases $\frac{1}{n_k - 1}$ second term I am making it $\frac{1}{n_k - 2}$. and so on and the last term is $\frac{1}{n_{k+1} + 1}$. So, this you can rewrite as $opt \sum_{i=1}^{n} \frac{1}{i}$ which is nothing, but the nth harmonic number.

$H_n \times opt$ ok. So, this proves the claim ok. So, let us stop here. Thank you.