

## Lecture 23: 4.14.05

Lecturer: Christos

Scribes: Michael Maire, Anil Sewani

**Disclaimer:** *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.*

## 23.1 Randomized Rounding Algorithm for SET-COVER

We start by formulating the SET-COVER problem as a Linear Program (LP). The LP for SET-COVER is based on minimizing the total cost of all sets in the solution subject to the constraint that each element in the universe  $U$  is covered. As discussed in the last lecture, this can be stated as follows:

$$\min \sum_{i=1}^k \text{cost}(S_i) \cdot x_i$$

subject to the constraints

$$\begin{aligned} \sum_{i:j \in S_i} x_i &\geq 1 \text{ for each } j \in U \\ x_i &\geq 0 \text{ for each } i = 1, 2, \dots, k \end{aligned}$$

The randomized rounding approximation algorithm for SET-COVER that uses the above LP formulation can be summarized as follows:

**Step 1:** Solve the LP, and treat the values of the LP solution  $x = (x_1, x_2, \dots, x_k)$  as a probability vector.

**Step 2:** Round the solution at random. Specifically, include the set  $S_i$  in the solution with probability  $x_i$ .

**Step 3:** Repeat the rounding in the previous step  $t$  times, and take the union of all covers obtained in the individual iterations. Call this result  $C$ .

Note that the above solution need not be a feasible one. We would like to obtain bounds on the probability that  $C$  is a feasible solution; *i.e.*, it is the cover of the universe  $U$ , for a fixed number of repetitions  $t$ . We would also like to obtain expected costs of our solution and bound the total cost of  $C$  in terms of the optimal cost obtained by solving the LP.

Lets start by considering a single iteration of Step 2. The probability that a single element  $j$  is not covered in  $C$  is:

$$\Pr[j \notin C] = \prod_{i:j \in S_i} (1 - x_i)$$

Let  $k_j$  be the number of sets to which  $j$  belongs. Since  $\sum_{i:j \in S_i} x_i \geq 1$ , it is clear that the above equation is maximized when  $x_i = \frac{1}{k_j}$ . Therefore, we get:

$$\Pr[j \notin C] = \prod_{i:j \in S_i} (1 - x_i) \leq \left(1 - \frac{1}{k_j}\right)^{k_j}$$

Recall that  $\lim_{k \rightarrow \infty} \left(1 - \frac{1}{k}\right)^k = \frac{1}{e}$ . Using this fact,

$$\Pr[j \notin C] = \prod_{i:j \in S_i} (1 - x_i) \leq \left(1 - \frac{1}{k_j}\right)^{k_j} \leq \frac{1}{e} \approx 36\%$$

Hence, the probability that an element is not covered in a single iteration of the randomized rounding algorithm for SET-COVER is  $\frac{1}{e}$  (and conversely, the probability that an element is covered in a single iteration is  $1 - \frac{1}{e}$ ).

Now, let's consider the case of multiple iterations of the Randomized Rounding algorithm presented above. For  $t$  iterations, the probability that an element  $j$  is not covered by  $C$  is  $\left(\frac{1}{e}\right)^t$ . This is because individual iterations are independent of one another.

We now consider the probability that  $C$  is not a proper set cover for  $U$  after  $t$  iterations. We use the "Union Bound" to get an upper bound on this probability. Recall that the Union Bound states that the probability of a union of several events is upper bounded by the sum of the individual event probabilities. Hence,

$$\Pr[E_1 \cup E_2 \cup \dots \cup E_N] \leq \sum_{i=1}^N \Pr[E_i]$$

Using this fact, we obtain the value of  $t$  for which the probability that  $C$  is not a proper cover of  $U$  is upper bounded by  $\frac{1}{4}$ .

$$\Pr[C \text{ is not a cover for } U] \leq \sum_{j \in U} \Pr[j \text{ is not covered}] \leq \frac{1}{4}$$

Denoting the total number of elements in  $U$  by  $n$ , we note that when  $t = \ln(4n)$ ,

$$\sum_{j \in U} \Pr[j \text{ is not covered}] = \sum_{j=1}^n \left(\frac{1}{e}\right)^{\ln(4n)} = n \left(\frac{1}{4n}\right) = \frac{1}{4}$$

Note that the expected cost of one iteration of the algorithm is  $\sum_{i=1}^k \text{cost}(S_i) \cdot x_i$ , which is the same as the optimal solution to the LP (which we will call  $OPT_f$ ). The expected cost of the randomized rounding solution  $C$  can hence be defined as

$$E[\text{cost}(C)] = \ln(4n) \cdot OPT_f$$

Recall that by Markov's inequality,

$$\Pr[X \geq k \cdot E[X]] \leq \frac{1}{k}$$

Using Markov's inequality and the expected cost of  $C$ , we can further say the following:

$$\Pr[\text{cost}(C) \geq 4 \ln(4n) \cdot \text{OPT}_f] \leq \frac{1}{4}$$

Thus, the probability that the algorithm provides a feasible solution within  $4 \ln(4n)$  of  $\text{OPT}_f$  can be denoted as:

$$\Pr[\text{algorithm succeeds}] \geq 1 - \frac{1}{4} - \frac{1}{4} = \frac{1}{2}$$

We can run the algorithm multiple times until it is successful. Note that we expect only to have to run it twice.

## 23.2 Hierarchy of Difficult Optimization Problems

As seen in previous lectures, we can currently group optimization problems into the following hierarchy (assuming  $P \neq NP$ ) based on the degree to which they can be approximated by a polynomial time algorithm.

Approximation Factor	Optimization Problems
no $\epsilon$	TSP, CLIQUE
$\epsilon = \log(n)$	SET COVER
some but not all $\epsilon > 0$	$\Delta$ TSP, MAXSAT
any $\epsilon > 0$	KNAPSACK
$\epsilon = 0$	all problems in P

How does one prove that approximation is NP-hard? For the Traveling Salesman Problem (TSP), we did this by creating a gap in the possible values of the objective function. Recall that we reduced the NP-complete Hamiltonian Cycle problem to the TSP optimization problem such that a “no” instance of Hamiltonian Cycle corresponded to a TSP with optimum at least  $(1 + \epsilon)L$ , but a “yes” instance corresponded to a TSP with optimum at most  $L$ , where  $L$  was a constant. Therefore, any polynomial time algorithm that approximated the TSP optimization problem within a factor of  $\epsilon$  could be used to solve an NP-complete decision problem. Assuming  $P \neq NP$ , no such algorithm exists.

In general, we create a gap by constructing a situation where if the answer is “yes”, something happens that is discontinuously away from that which happens if the answer is “no”. For a long time, the above theorem about TSP was the only gap theorem known.

While creating gaps may be difficult, there is a known method for propagating gaps. In particular, there are a set of careful reductions starting from MAXSAT that preserve gaps. For example, there are gap-preserving reductions from MAXSAT to CLIQUE,  $\Delta$ TSP, and VERTEX COVER. However, no gap theorem was known for MAXSAT until the development of PCP theory.

## 23.3 PCP Theory

Recall the definition of NP. A problem  $A \in NP$  iff there is a polynomial-time verifier  $V$  such that for input  $x$ , if  $x$  is a “yes” instance, there exists a polynomial-size certificate  $c$  such that  $V(c)$  outputs “yes”, and if  $x$  is a “no” instance, no such certificate exists ( $V(c)$  outputs “no” on all  $c$ ).

Does this verifier have to be polynomial time?

Cook showed that NP has extremely simple verifiers. All problems in NP can be reduced to 3SAT and 3SAT has a log space verifier. In particular, when checking the satisfiability of each clause, we need only use  $\log(n)$  additional space when working on an input of size  $n$ .

PCP theory takes the reduction from polynomial verifiers to Cook verifiers one step further, to PCP verifiers. PCP stands for Probabilistically Checkable Proof.

A PCP verifier  $V$  operates on a certificate  $c$  as follows. First, we randomly flip  $O(\log(n))$  coins, where  $n$  is the size of our problem instance. These coins are used to select  $q$  bits of  $c$ , where  $q$  is a fixed constant. We feed these  $q$  bits to  $V$  as input.  $V$  must satisfy the following two properties.

If  $x$  is a “yes” instance, then  $\exists c$  such that  $\text{Prob}(V \text{ accepts } c) = 1$ .

If  $x$  is a “no” instance, then  $\forall c$ ,  $\text{Prob}(V \text{ rejects } c) \geq \frac{1}{2}$ .

**Theorem 23.1** *NP has PCP verifiers.*

How is this achieved? The actual proof is long and complicated, so what follows is only a sketch. The “yes” part is easy. Simply accept as long as the few bits that are checked pass. For the “no” part, we want our verifier to be able to detect lies (bits that would indicate the certificate was invalid) even though it only looks at a few bits. The key idea is to use coding theory to spread any lie in a certificate regularly throughout the entire certificate. The verifier will then either catch the lie or catch the lack of required regularity in the certificate.

We can use this fact to prove a gap theorem for MAXSAT.

**Theorem 23.2** *For some  $\epsilon > 0$ , MAXSAT cannot be approximated better than  $(1 - \epsilon)$ .*

**Proof:** Let  $V$  be a PCP verifier for some NP-complete problem. In particular, for inputs of length  $n$ , verification works by flipping  $k \log(n)$  coins in order to choose  $q$  bits of the certificate  $c$  as input to  $V$ , where  $q$  is a constant. Furthermore, regard  $V$  as a circuit with  $q$  inputs.

There are  $2^{k \log(n)} = n^k$  possible coin flips. So there are  $n^k$  possible circuits  $V$ . Create one variable for each bit position in the certificate  $c$ . We can rewrite each circuit as a 3SAT formula involving exactly  $q$  of these variables. As  $q$  is a constant, each of these formulas is of constant size, say at most  $D$  clauses.

Now consider the MAXSAT problem that is given as input the set of  $n^k D$  clauses corresponding to all of these circuits.

For a “yes” instance of the original NP-complete problem, there exists a certificate satisfying all of the verifiers. So all  $n^k D$  clauses can be satisfied.

For a “no” instance of the original NP-complete problem, any certificate must be rejected by at least half of the verifiers. So any assignment of values to the certificate bits must leave at least one unsatisfied clause in at least  $\frac{n^k}{2}$  of the verifiers. So at most  $n^k D - \frac{n^k}{2}$  clauses can be satisfied.

This is a gap for MAXSAT with  $\epsilon = \frac{1}{2D}$ .

■