Computational Complexity November 22, 2018

Lecture 13 (Notes)

Lecturer: Ola Svensson *Scribes: Ola Svensson*

Disclaimer: These notes were written for the lecturer only and may contain inconsistent notation, typos, and they do not cite relevant works. They also contain extracts from the two main inspirations of this course:

1. The book *Computational Complexity: A Modern Approach* by Sanjeev Arora and Boaz Barak;
2. The course *http://theory.stanford.edu/ trevisan/cs254-14/index.html* by Luca Trevisan.

1 Introduction

Recall last lecture:

- Probabilistic Checkable Proofs. **PCP:** A language L is in PCP(r, q) if there exists a probabilistic polynomial time verifier V which, when given an instance x and a proof Π of length $poly(|x|)$, proceeds as follows
  1. Reads x and $O(r)$ random bits.
  2. Based on x and $O(r)$ random bits, queries $O(q)$ bits from Π.
  3. Computes and either accepts or rejects with following properties,
     - **completeness:** If $x \in L$ then $\exists$ proof Π such that $V^{\Pi}(x)$ accepts with probability 1.
     - **Soundness:** If $x \notin L$ the $\forall$ proof Π, the verifier $V^{\Pi}(x)$ accepts with probability at most $\frac{1}{2}$.
- PCP-Theorem $\mathbf{NP} = \mathbf{PCP}(\log n, 1)$.
- Proof that $\mathbf{NP} \subseteq \mathbf{PCP}(poly(n), 1)$ by giving a $\mathbf{PCP}(poly(n), 1)$ verifier for QUADEQ.

Today:

- Connection between PCP-Theorem and (hardness of) approximation algorithms.
- Clique is hard to approximate.

2 Hardness of Approximation

We now explore the fantastic implications the PCP-Theorem has had on our understanding of approximation algorithms. Indeed, the PCP-Theorem shows that Max-3SAT is hard to approximate and then using that as a starting point you can prove your favorite problem to be hard to approximate. The PCP-Theorem has had a similar impact on our understanding of the complexity of approximation as the Cook-Levin Theorem had on our understanding of the limits of exact algorithms.

2.1 Definition of approximation algorithms

Definition 1 *An α-approximation algorithm for a given optimization problem is an algorithm which can be run in polynomial time and which outputs a solution S such that:*

- $\frac{cost(S)}{cost(Optimal\ solution)} \leq \alpha$ *if the problem is a minimization problem ;*

- $\frac{profit(S)}{profit(Optimal\ solution)} \geq \alpha$ *if the problem is a maximization problem.*

It is clear that we will have $\alpha > 1$ for minimization problems and $\alpha < 1$ for maximization problems.

It is possible to prove hardness of approximation results without using the PCP-theorem, as we are going to see in the following example.

2.2 Hardness of traveling salesman problem (TSP)

An instance of TSP is a graph where each edge is labeled with a cost, that must be a positive or a null real number. The expected output is a Hamiltonian path of minimal total cost in this graph.

What we are going to prove is that TSP falls into the worse category of approximation, that is:

Theorem 2 *For any $\alpha \geq 1$, it is NP-hard to get an α-approximation of TSP.*

Proof To prove this, we are going to introduce HAM, which is the problem of determining whether a graph has an hamiltonian cycle. We will use the fact that HAM is NP-hard (we can notice that this directly implies that TSP is NP-hard, since TSP asks for an Hamiltonian cycle in the graph). What we need is a **gap-introducing reduction** from HAM to TSP, that is, a reduction from HAM to TSP. Namely, from a graph G given as input to HAM, it builds a graph G' such that:

- if G has an Hamiltonian cycle, then G' has an Hamiltonian cycle with "low" cost.
- if G has non Hamiltonian cycle, then all Hamiltonian cycles of G' have "high" cost.

The main idea is that knowing a sufficiently good approximation of the solution of TSP for an input that we know to be in either the "high cost" or "low cost" category will enable us to determine in which of the two categories it belongs. This means that a good approximation for TSP can be used to solve HAM because it can distinguish between the two kind of outputs of our reduction.

To do this, we choose a constant M (that we will want to be very high) and we define our reduction as follows:

- The set of nodes of G' is equal to the set of nodes of G.
- G' is complete (all nodes are connected by an edge)
- The weight of an edge e of G' is 1 if e is also an edge of G, and is M otherwise

We denote by n the number of nodes in G and G'. If there exists a Hamiltonian cycle in G, then the same cycle has cost n in G'. Otherwise, any Hamiltonian cycles in G' use at least one edge that does not exist in G, thus it has a cost greater than M.

Now, suppose that we have an $\frac{M}{n}$-approximation of TSP. We denote by S the solution given by our approximation with input G'. If G has a Hamiltonian cycle, then the optimal solution has cost n. As such the cost of S is less than M. Otherwise, the optimal solution has cost greater than M, which means that S also has a cost greater than M. Therefore, we can decide whether G had a Hamiltonian cycle by looking at the cost of S, so our algorithm solves HAM. If M can be arbitrary, we can choose it as large as we want, which proves our theorem. ■

Observation 3 *Although we did not need the PCP-theorem to conclude, we were somehow "lucky" to be able to manipulate the costs of the edges at will. This trick is not a general method since many NP-hard problems, such as looking for the largest clique in a graph are purely combinatorial.*

Observation 4 *We can sometimes get positive results about existence of approximations. For instance, consider the modified version of TSP where the costs along the edges are required to satisfy the triangular inequality, i.e., for $u, v, w \in V$, $cost(u, w) \leq cost(u, v) + cost(v, w)$. The convention is that the cost between two nodes is infinite if there are no edges between them. This problem remains NP-hard. However, a well-known linear 2-approximation consists in performing a depth-first traversal of a minimum spanning tree of the graph (a sub-graph of G, which is a tree and which contains all nodes of G, and for which the sum of the costs of all edges is minimal among all spanning tree of G). A polynomial 1.5-approximation have also been found, and the theoretical limit has not been reached yet.*

2.3 Maximize accept probability

PCP verifiers, themselves, are a source of NP-hard problems. Given a $\mathbf{PCP}(\log(n), 1)$ verifier V for SAT, and an input φ, one can wonder which proof has a maximal probability of being accepted by V along with input φ.

Theorem 5 *It is NP-hard to approximate this problem within a factor of $\frac{1}{2}$.*

Proof The proof of this assertion is very simple, because the gap we need has already been introduced by the definition of a PCP verifier. Let's assume that we have a $\frac{1}{2}$-approximation for this problem. Let Π be the proof constructed by our algorithm with input φ, and p be the probability that Π is accepted by V along with input φ. We can evaluate p in polynomial time by feeding V with all possible random strings since it asks only for $O(\log(n))$ random bits. If $p \geq \frac{1}{2}$, then by the soundness property of V, φ is satisfiable. Otherwise, we have $p < \frac{1}{2}$ which means that the optimal proof cannot have a probability greater than $2p$ of being accepted, and $2p < 1$. By the completeness property of V, φ cannot be satisfiable.

Therefore, running our approximation algorithm on φ, along with an additional polynomial computation tells us whether φ is satisfiable, which means that our verifier solves SAT. ■

2.4 Hardness of Clique

Here we study the hardness of approximation of the maximum clique problem (or simply the clique problem). The clique problem consists in finding a maximum cardinality clique (i.e., a subset of vertices which forms a complete graph) given an undirected graph $G(V, E)$. We show here that there exists a constant $\epsilon > 0$, such that it is NP-hard to approximate the clique problem within a factor of $\frac{1}{N^\epsilon}$. We start by studying a gap introducing reduction from the SAT problem to a graph which is used to argue on the hardness results.

Lemma 6 *For fixed constants b and q, there is a gap introducing reduction that transforms, in polynomial time, a SAT formula φ of size n to a graph $G(V, E)$, where $N = |V| = n^b 2^q$ such that;*

- ***Completeness**: If φ is satisfiable, $OPT(G) \geq n^b$.*
- ***Soundness**: If φ is not satisfiable, $OPT(G) < \frac{n^b}{2}$.*

Proof Let F be a $\mathbf{PCP}(\log n, 1)$ verifier for SAT that requires $b \log n$ random bits and reads q bits from the proof.
We transform SAT instance φ into G[1] as follows.

- **Vertices**: For each couple of random string r of $b \log n$ bits and truth assignment τ for q boolean variable, there is a vertex $V_{r,\tau}$ in G.

[1] This graph is known as FGLSS graph

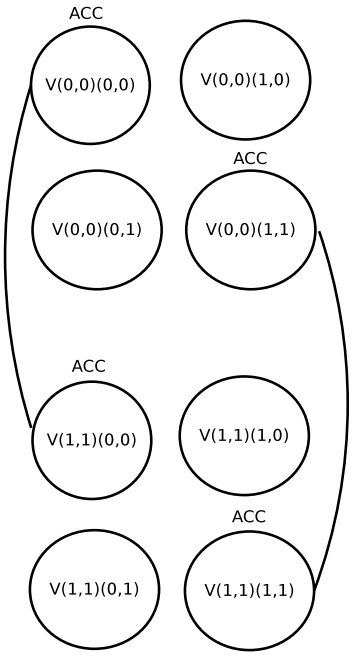

Figure 1: FGLSS graph for $b = 2$ and $q = 2$ (only a subset of the vertices and edges are drawn).

- **Edges**: Let $Q(r)$ represent the q positions in the proof that F queries given the "random string" r. Now, two vertices V_{r_1,τ_1} and V_{r_2,τ_2} are *consistent*, if τ_1, τ_2 agree on the values of the proof where $Q(r_1)$ and $Q(r_2)$ overlap. Further, we say that $V_{r,\tau}$ is *accepting* if F accepts, given a "random string" r and read τ in the proof. If two vertices are both *consistent* and *accepting*, there is an edge between them.

For example, consider the FGLSS graph defined for 2 random bits and 2 query strings illustrated in Figure 1. Vertices in the graph are labeled as $V(randombits)(querybits)$. Edges thus correspond to the vertices which are both consistent and accepting.
We will argue on the completeness and soundness based on the FGLSS graph.

- **Completeness**: According to the definition, we know that if φ is satisfiable there exists a proof Π such that F accepts with probability 1. Let $S = \{V_{r,\tau} \mid \tau \textit{ consistent with } \Pi\}$. By the definition of the transformation, for every "random string" r, there exists one vertex $V_{r,\tau} \in S$. Therefore, the cardinality of the set S becomes the number of random bits. n^b. We can see that S forms a clique since for each $V_{r_i,\tau_i}, V_{r_j,\tau_j} \in S$ they are accepting vertices as well as consistent (due to the definition of S).

- **Soundness**: Again by the definition of the verifier, if φ is not satisfiable, for all the proofs, F accepts with probability $< \frac{1}{2}$.

  Let C be a clique in G. Since all pairs of vertices in C are consistent, we know that C contains at most one vertex associated with each random string. This also implies that it is possible to build a proof Π that is consistent with every vertex of C: each vertex of C forces the value of up to k bits in the proof but the fact that the vertices are consistent ensures that there will be no contradiction. Some bits of Π might not be specified by any vertex, i.e., we can freely decide regardless of their value. For instance, we can set them to 0.

Now, we can observe that Π is accepted with probability at least $\frac{|C|}{n^b}$, because each vertex in C corresponds to a different random string that will make our verifier accept C. Because of the soundness of the verifier, this proves that $|C| < \frac{n^b}{2}$.

■

3 Improved hardness of approximating the Clique problem

In this section, we first make a simple adaptation of the previous reduction so as to show that Clique has no constant-factor approximation algorithm. We then use "expanders" to get polynomial hardness.

3.1 No constant approximation guarantee

We observe that the techniques in Section 2.4, imply the following stronger result:

Lemma 7 *Show that it is NP-hard to approximate the Clique problem within any constant factor.*

Proof The reduction in Section 2.4 was so that the graph G constructed from a SAT formula φ satisfied:

- **Completeness**: If φ is satisfiable, $OPT(G) \geq n^b$.
- **Soundness**: If φ is not satisfiable, $OPT(G) < \dfrac{n^b}{\rho}$,

where ρ is a parameter set to $1/2$ in that section. But why was ρ set to $1/2$? That came from that the soundness of the PCP-verifier of SAT. But by repeating the verifier check any *constant* number of times, say s times, we will still have a $\mathbf{PCP}(\log n, 1)$ verifier for SAT but now with soundness $1/2^s$. Specifically, if the original verifier used $b \log n$ random bits and queried q bits from the proof then the verifier with improved soundness uses $sb \log n$ random bits and reads sq bits from the proof.

For any constant s, the reduction thus runs in polynomial time. This implies that Clique is NP-hard to approximate within a factor $1/\rho$ for any constant ρ. ■

3.2 Polynomial Hardness of Clique

We shall show the following theorem.

Theorem 8 *For some constant $\epsilon > 0$, it is NP-hard to approximate clique within a factor $1/N^\epsilon$ on a graph with N vertices.*

Note that in Lemma 7 we used the fact that Lemma 6, when given a verifier V that uses $b \log n$ random bits and q bits from the proof and soundness 2^s, says that clique is hard to approximate within a factor $1/2^s$. Moreover, the reduction runs in time $n^b 2^q$ and naively $b = O(s)$ and $q = O(s)$. Therefore, it seems like s can only be chosen to be a constant to have a polynomial time reduction. However, note that the reduction is still polynomial if $q = O(\log n)$ and $b = O(1)$. Therefore, a verifier V for SAT that reads $b' \log n$ random bits and $q' \log n$ bits from the proof with soundness $1/n$ would imply that Clique is NP-hard to approximate within a factor $1/n$. Now using that $N = n^{b'} 2^{q' \log n} = n^{1/\epsilon}$ for some constant $\epsilon > 0$ yields the theorem.

It remains to prove that such a verifier V exists for SAT, i.e., we wish to save random bits :).

- Let F be $\mathbf{PCP}(\log n, 1)$ for SAT which queries $b \log n$ random bits and q bits from the proof.

- We shall show how to obtain V from F.
- Let H be a constant degree expander graph on n^b vertices each vertex having a unique $b \log n$ bit label (one for each possible random string of F). Many of you have not seen expanders before. Informally it is sparse (constant degree) graph that almost behaves as the complete graph with respect to cuts and random walks. One definition is as follows: a d-regular graph on vertex set U is an expander if for every $S \subseteq U : |S| \leq |V|/2$ the following holds: take a random vertex u of S and a random neighbor v of u then $v \notin S$ with some fixed constant probability. Note that a complete graph satisfies this with probability $1/2$.

  The main property of expanders we will use here is:

  Lemma 9 *Let S be any subset of vertices of H consisting of at most half the vertices. Then there is a constant k such that*

  $$\Pr[\textit{all vertices of a } k \log n \textit{ random walk lie in } S] < \frac{1}{n}.$$

- Do a random walk on H of length $k \log n$ using $O(\log n)$ $(q' \log n)$ random bits.
  - The label of each vertex on this path specifies a $b \log n$ random bit string. It uses these $k \log n + 1$ string as the "random" strings on which it simulates the verifier F.
  - V accepts iff F accepts on all $k \log n + 1$ runs.

Completeness Suppose the verifier is given a formula φ that is satisfiable. Then there exists a proof π that makes F always accept; since V accepts if F accepts on all runs, V accepts this proof with probability 1 as well.

Soundness Suppose the verifier is given a formula φ that is not satisfiable. Then no matter which proof π we consider, verifier F accepts with probability $< 1/2$. Let S denote the set of vertices of H that correspond to random strings on which F accepts. Then $|S| < n^b/2$. Now V accepts iff the random walk remains entirely in S. It is known that there exists a constant k such that

$$\Pr[\text{all vertices of a } k \log n \text{ length random walk lie in } S] < 1/n.$$