CMPUT 675: Approximation Algorithms **Fall 2014**

Lecture 30 (Nov 21 & 24): Introduction to Hardness

Lecturer: Zachary Friggstad *Scribe: Zachary Friggstad*

30.1 Introduction by Example: Hardness of Max-3SAT

We have seen a few hardness results in earlier lectures, but those techniques are not strong enough to establish hardness of approximation results for a number of problems including Minimum Vertex Cover, MAX-SAT, MAX-Cut, etc. The next few lectures will discuss some techniques for proving such lower bounds. We begin with an example.

Definition 1 *In* Max-2Lin(3)*, we are given variables $x_1, \ldots, x_n$ over $\mathbb{Z}/2\mathbb{Z}$ (i.e. integers mod 2). We are also given linear constraints of the form $x_i + x_j + x_k \equiv b \pmod 2$ where $1 \leq i < j < k \leq n$ and $b \in \{0, 1\}$. The goal is to find an variable assignment that maximizes the number of satisfied constraints.*

A random assignment will satisfy each constraint with probability exactly $1/2$, so this is a simple $1/2$-approximation (and it can be derandomized easily). This is essentially the best approximation.

Theorem 1 (Håstad [H01]) *For every constant $\epsilon > 0$ and every language $L \in \mathbf{NP}$, there is a polynomial-time reduction from L to* Max-2Lin(3) *such that:*

- Completeness*: A* yes *instance of L is mapped to an instance of* Max-2Lin(3) *where a $(1-\epsilon)$-fraction of clauses can be satisfied by some solution.*
- Soundness*: A* no *instance of L is mapped to an instance of* Max-2Lin(3) *such that any solution satisfies at most a $(1/2+\epsilon)$-fraction of clauses.*

Here, the ϵ factors into the running time of the reduction (in this case, the running time of the reduction is $n^{O(1/\epsilon)}$). Reductions of this sort that produce a *gap* between *yes* and *no* instances are helpful in establishing hardness.

Corollary 1 *For any constant $\epsilon > 0$, there is no $(1/2+\epsilon)$-approximation for* Max-2Lin(3) *unless $\mathbf{P} = \mathbf{NP}$.*

Proof. Suppose, for some constant $c > 1/2$, there was a c-approximation algorithm $\mathcal{A}$ for Max-2Lin(3). We will use $\mathcal{A}$ to solve any decision problem in **NP** in polynomial time.

Choose $\epsilon > 0$ such that $(1-\epsilon) \cdot c > 1/2 + \epsilon$ (choosing $\epsilon = (c - 1/2)/2$ suffices). Reduce an instance of L to Max-2Lin(3) using the given reduction (with this ϵ) and then approximation the optimum solution of this Max-2Lin(3) instance using $\mathcal{A}$. Declare that the instance of L is a *yes* instance if and only if $\mathcal{A}$ satisfies more than a $(1/2+\epsilon)$-fraction of the constraints.

If we started with a *yes* instance of L, then this will find a solution where the fraction of satisfied clauses is at least $c \cdot (1-\epsilon) > (1/2+\epsilon)$ so we correctly identify it is a yes instance. If we started with a *no* instance of L, then $\mathcal{A}$ cannot satisfy more than a $(1/2+\epsilon)$-fraction of clauses so we correctly identify it is a no instance. Thus, this algorithm decides L in polynomial time. Since L was an arbitrary language in **NP**, we can use this technique to solve any problem in **NP** in polynomial time. ■

30.1.1 General Discussion of Gap Reductions

This is the general recipe we will follow in the coming lectures. Suppose we can reduce any language $L \in \mathbf{NP}$ (or even just a single **NP**-complete language) to the maximization problem $\mathcal{P}$ we want to show hardness for such that:

- Completeness: starting with a *yes* instance of L produces an instance of $\mathcal{P}$ with optimum value at least α.
- Soundness: starting with a *no* instance of L produces an instance of $\mathcal{P}$ with optimum value at most β.

This approach also works for minimization problems if we ensure where a yes instance of L is reduced an instance of $\mathcal{P}$ with value *at most* α and a no instance of L is reduced an instance of $\mathcal{P}$ with value *at least* β.

We say that such a reduction demonstrates a *hardness gap* of α vs. β. If a problem has a hardness gap of α vs. β, then we cannot approximate the problem better than β/α unless $\mathbf{P} = \mathbf{NP}$.

The most common way to prove a (maximization) problem $\mathcal{P}$ has a hardness gap of α vs. β for some values α, β is to take a (maximization) problem $\mathcal{P}'$ we already know a hardness gap of α' vs. β' and reduce to the problem $\mathcal{P}$. This should be done so that an instance of $\mathcal{P}'$ with optimum value at least α' maps to an instance of $\mathcal{P}$ with optimum value at least α (i.e. the completeness is roughly preserved), and an instance of $\mathcal{P}'$ with optimum value at most β' maps to an instance of $\mathcal{P}$ with optimum value at most β (i.e. the soundness is roughly preserved).

If either $\mathcal{P}$ or $\mathcal{P}'$ are minimization problems, then essentially the same strategy works except we replace "at least" and "at most" in the appropriate places.

30.1.2 The Gap Reduction for Max-3SAT

Let's see this technique action for Max-3SAT.

Theorem 2 (Håstad [H01]) *For any constant $\epsilon > 0$, there is no $(7/8 + \epsilon)$-approximation for* Max-3SAT *unless* $\mathbf{P} = \mathbf{NP}$.

We already saw a 7/8-approximation for instances that contain exactly three distinct variables per clause: a random assignment satisfies each clause with probability exactly 7/8. This reduction will produce instances with exactly three variables per clause, so this naive algorithm cannot be improved by any constant factor.

Proof. We reduce from Max-2Lin(3) to demonstrate a hardness gap of $1 - \epsilon$ vs. $\frac{7}{8} + \epsilon$. Consider an instance of Max-2Lin(3) with variables $x_1, \ldots, x_n$. We use the same variables in the Max-3SAT instance. It will be helpful to think $0 \equiv$ False and $1 \equiv$ True.

Consider a constraint of the form $x_i + x_j + x_k \equiv 0 \pmod 2$ of Max-2Lin(3). We create four new clauses in the Max-3SAT instance that we refer to as a *4-clause group*. The clauses are:

$$\bar{x}_i \vee \bar{x}_j \vee \bar{x}_k, \qquad \bar{x}_i \vee x_j \vee x_k \qquad x_i \vee \bar{x}_j \vee x_k \qquad x_i \vee x_j \vee \bar{x}_k.$$

Similarly, for a constraint of the form $x_i + x_j + x_k \equiv 1 \pmod 2$ we create the following 4-clause group:

$$x_i \vee x_j \vee x_k \qquad x_i \vee \bar{x}_j \vee \bar{x}_k \qquad \bar{x}_i \vee x_j \vee \bar{x}_k \qquad \bar{x}_i \vee \bar{x}_j \vee x_k.$$

This completes the description of the reduction.

Observation: A particular setting of variables x satisfies this constraint of the Max-2Lin(3) instance if and only if all four of the clauses in the corresponding 4-clause group are satisfied.

Now we invoke Theorem 1 with a small enough constant $\epsilon' > 0$ (that we specify later) to reduce any language in **NP** to Max-3SAT by first using the reduction to Max-2Lin(3) in the statement of the theorem, and then by reducing this instance to a Max-3SAT instance using the reduction described above. Let m be the number of constraints in the Max-2Lin(3) instance, so $4m$ is the number of clauses in this Max-3SAT instance.

If we started with a *yes* instance, then the number of constraints that can be satisfied in this Max-3SAT instance is at least $4 \cdot (1-\epsilon') \cdot m$ by using the same assignment that satisfies at least $(1-\epsilon') \cdot m$ of the constraints in the Max-2Lin(3) instance.

Now for the soundness, assume that the instance of L was a *no* instance. Consider any truth assignment x for the Max-3SAT instance and say that the number of 4-clause groups that have all four clauses satisfied is $\alpha \cdot m$. By the above observation, this x satisfies $\alpha \cdot m$ constraints of the Max-2Lin(3) instance so, by Theorem 1, we have $\alpha \leq 1/2 + \epsilon'$.

The number of clauses satisfied by x can be bounded as follows:

$$\begin{aligned} 4 \cdot \alpha \cdot m + 3 \cdot (1-\alpha) \cdot m &= (3+\alpha) \cdot m \\ &\leq \left(3 + \frac{1}{2} + \epsilon'\right) \cdot m \\ &= \left(\frac{7}{8} + \frac{\epsilon'}{4}\right) \cdot 4 \cdot m. \end{aligned}$$

To summarize, if we started with a yes instance of L then we can satisfy a $(1-\epsilon')$-fraction of constraints and if we started with a no instance of L then we can satisfy at most a $(7/8 + \epsilon'/4)$-fraction. For any given constant $\epsilon > 0$, we can choose $\epsilon' > 0$ small enough so that

$$\frac{\frac{7}{8} + \frac{\epsilon'}{4}}{1-\epsilon'} < \frac{7}{8} + \epsilon.$$

This establishes the hardness of approximating Max-3SAT with an approximation guarantee of $7/8 + \epsilon$. ■

30.2 The PCP Theorem

The techniques above demonstrate how we can reduce from a problem with a *hardness gap* to another problem. However, it assumes that there is already an appropriate problem to reduce from that has a sufficiently large hardness gap.

There must be an original problem with a *hardness gap* that we can use to start this process, but where does it come from? This is analogous to the theory of **NP**-completeness you likely saw in your algorithms courses: we can reduce an **NP**-complete problem to another problem in **NP** to prove that it is also **NP**-complete, but where does the first **NP**-complete problem come from?

In the case of **NP**-completeness, the Cook-Levin theorem gives us an initial NP-complete problem: SAT. However, a careful examination of the proof you likely saw for this shows that even in the *no* case, it is still possible to satisfy most clauses (probably even all except one last clause which asserts the final state is the accepting state)! We need a better starting point to begin demonstrating hardness gaps.

The answer for hardness gaps is the *PCP Theorem* (discussed below), which can be seen as a robust version of the Cook-Levin theorem. We will use it to produce an instance of Max-SAT that has a constant hardness gap. This will show that there is no c-approximation for Max-SAT for some constant $c < 1$. Getting optimal constants requires a lot more work, but at least this shows there is no PTAS for Max-SAT.

30.2.1 Verifiers for NP

One way to define the class **NP** is to discuss it in terms of verifiers. Informally, you know **NP** to be the class of all languages where a proof of membership can be efficiently checked. For example, we can efficiently check that a proposed Hamiltonian cycle is indeed a Hamiltonian cycle, so the language L of Hamiltonian graphs is in NP.

More formally, we say that $L \in \mathbf{NP}$ if there is a Turing Maching V that reads two inputs x, π. Here, x is an instance of the decision problem $x \in L$? and π is a proposed proof of this statement. Then $V(x, \pi)$ runs in $\text{poly}(|x|)$ time and decides whether to accept the proof π of the statement $x \in L$ or not. Furthermore, V has the following properties for an input x:

- Completeness: If $x \in L$, then there is some proof string π such that $V(x, \pi) = \text{ACCEPT}$.
- Soundness: If $x \notin L$, then $V(x, \pi) = \text{REJECT}$ for every proof string π.

Now consider the following alternative verifier.

Definition 2 *An $(r(n), q(n))$-restricted verifier for a language L with completeness c and soundness s is a polynomial-time Turing Machine V that reads three inputs x, b, π, where we think of x and the input string, b as the random bit string, and π as the proof string. The verifier V has the following properties:*

- *V has random access to π.*
- *V runs in $\text{poly}(|x|)$ time, reads only the first $r(|x|)$ bits of b, and only $q(|x|)$ bits of π (which can be any of the bits of π, not just the first $q(|x|)$).*
- *V queries π non-adaptively: it reads the random bit string b and decides which $q(n)$ bits of π to query before looking any of them.*
- Completeness*: If $x \in L$ then there is some proof string π such that $\Pr_{b \sim \{0,1\}^{r(|x|)}}[V(x, b, \pi) = \text{ACCEPT}] \geq c$.*
- Soundness*: If $x \notin L$ then $\Pr_{b \sim \{0,1\}^{r(|x|)}}[V(x, b, \pi) = \text{ACCEPT}] \leq s$ for every proof string π.*

Here, the probability is over choosing the random bit string b uniformly from $\{0,1\}^{r(|x|)}$.

So, what languages have such a verifier? It depends on the parameters $r(n), q(n), c, s$, and we define a class of languages for each such setting of parameters. Here, the term **PCP** stands for "probabilistically checkable proof".

Definition 3 $\mathbf{PCP}_{c,s}(r(n), q(n))$ *is the class of all languages L that have an $(r(n), q(n)$-restricted verifier with completeness c and soundness s.*

Sometimes the completeness and soundness parameters are omitted from the definition, in which case we understand that $\mathbf{PCP}(r(n), q(n)) = \mathbf{PCP}_{1,1/2}(r(n), q(n))$.

For example:

- $\mathbf{PCP}_{1,0}(0, 0) = \mathbf{P}$
- $\mathbf{PCP}_{1,0}(0, \text{poly}(n)) = \mathbf{NP}$

- $\mathbf{PCP}_{1/2,0}(\text{poly}(n), 0) = \mathbf{RP}$ (the class of languages that can be solved with a poly-time randomized algorithm with one-sided error)

The major result that will be our starting point for gap reductions is the following celebrated theorem.

Theorem 3 (The PCP Theorem [AS98,A+98]) $\mathbf{PCP}_{1,1/2}(r \cdot \log_2 n, q) = \mathbf{NP}$ *for some constants* r, q.

Sometimes this is stated simply as $\mathbf{PCP}(O(\log n), O(1)) = \mathbf{NP}$.

The remarkable part of this statement is that we can write a proof for any language in **NP** such that we only need to check a *constant* number of bits of the proof to decide if it is valid or not! The only sacrifice is that a no instance will only be rejected with some constant probability. The choice of the soundness constant of $1/2$ is a bit arbitrary; we can decrease it by simply running V multiple times on the same proof π. In particular, for any $k \geq 1$ we have $\mathbf{NP} = \mathbf{PCP}_{1,1/2^k}(k \cdot r \cdot \log_2 n, k \cdot q)$.

Unfortunately, we do not have time to see a proof of the PCP theorem in our lectures. One direction, namely $\mathbf{PCP}_{1,1/2}(r \cdot \log_2 n, q) \subseteq \mathbf{NP}$, is somewhat simple and will be discussed it briefly below. However, the other direction is much more difficult (and **much** more interesting!). A full proof can be found in the book by Arora and Barak [AB09] mentioned on the class webpage.

If anyone is curious, I think it would be fun running a few seminars to discuss the proof! :)

30.3 Hardness from PCPs

We show how to use the PCP theorem (Theorem 3) to get a hardness gap in SAT.

Theorem 4 *There is a constant* $q \geq 1$ *such that for any language* $L \in \mathbf{NP}$ *there is a reduction from* L *to* Max-qSAT *such that:*

- *Completeness: If the instance of* L *is a* yes *instance, then the* Max-qSAT *instance is satisfiable.*
- *Soundness: If the instance of* L *is a* no *instance, then every truth assignment for the* Max-qSAT *instance satisfies at most a* $\left(1 - \frac{1}{2^{q+1}}\right)$*-fraction of satisfied clauses.*

Thus, we cannot approximate Max-qSAT within a ratio better than $1 - 2^{-q-1}$, which is close to the $1 - 2^{-q}$ approximation we get with the random truth assignment for instances with exactly q variables per clause.

Proof. Let V be an $(r \cdot \log_2 n, q)$-restricted verifier for L with completeness 1 and soundness $1/2$ (c.f. Theorem 3). Let x be an instance of the decision problem $x \in L$? and say $n = |x|$.

We have one variable in the Max-qSAT for each index into the proof string. Since there are $2^{r \cdot \log_2 n} = n^r$ random bit strings and since only q bits are queried, then there are at most $q \cdot n^r = \text{poly}(n)$ variables. We will think of a particular proof string as a setting of values to the variables of the Max-qSAT instance and vice-versa.

For every $b \in \{0,1\}^{r \cdot \log_2 n}$, we simulate V given random bit string b to determine the indices $i_1^b, \ldots, i_q^b$ of π that will be queried. Consider the boolean function $f^b : \{0,1\}^q \to \{0,1\}$ over variables $\pi^b = (\pi_{i_1^b}, \ldots, \pi_{i_q^b})$ where $f^b(\pi^b) = 1$ if and only if V, when given random string b, accepts a proof π that agrees with π^b on the queried bits.

It is a standard exercise from digital logic design to show that f^b can be represented by a q-CNF formula. For example, if $q = 3$ and $(0,1,0),(1,1,0)$ are the only settings of values to π^b that cause V to reject on random string b, then the clauses in this q-CNF representing f^b are:

$$\pi_{i_1^b} \vee \overline{\pi}_{i_2^b} \vee \pi_{i_3^b} \qquad \text{and} \qquad \overline{\pi}_{i_1^b} \vee \overline{\pi}_{i_2^b} \vee \pi_{i_3^b}$$

Let $C(b)$ be the clauses in the q-CNF representation of f^b. The final instance consist of clauses $\bigcup_{b \in \{0,1\}^{r \cdot \log_2 n}} C(b)$ (this is a disjoint union, keep all copies of each clause). Let Φ denote this q-CNF and let $m = \sum_b |C(b)|$ denote the number of clauses in Φ.

Since $|C(b)| \leq 2^q$ for each b, then $m \leq 2^q \cdot n^r$, which is polynomial. Furthermore, computing $C(b)$ for a given b requires only 2^q simulations of V, each of which takes polynomial time, so this construction of Φ takes polynomial time.

Important Observation
For an assignment/proof string π and for each $b \in \{0,1\}^{r \cdot \log_2 n}$, all clauses in $C(b)$ are satisfied by π if and only if $V(x,b,\pi)$ accepts. This is simply by construction of the clause sets $C(b)$.

Completeness
If x was a *yes* instance (i.e $x \in L$) then there is some π such that $V(x,b,\pi)$ accepts for every $b \in \{0,1\}^{r \cdot \log_2 n}$. This means using π as the truth assignment for Φ causes every clause in every $C(b)$ to be satisfied, so all m clauses of Φ are satisfied.

Soundness
Now suppose that x is a *no* instance. This means $V(x,b,\pi)$ rejects every proof π for at least half of the random bit strings $b \in \{0,1\}^{r \cdot \log_2 n}$.

Consider a truth assignment π for Φ and let $\alpha \cdot n^r$ be the number of b such that all clauses in $C(b)$ are satisfied by π. This means the number of satisfied clauses is at most $m - (1-\alpha) \cdot n^r$. Since V accepts π with probability $\leq 1/2$ and since there are n^r possible random strings, then the above observation means $\alpha \leq 1/2$.

We observed above that $m \leq 2^q \cdot |C(b)| = 2^q \cdot n^r$. Thus, the number of satisfied clauses is at most

$$m - (1-\alpha) \cdot n^r \leq m - n^r/2 \leq m - \frac{m}{2^{q+1}} = \left(1 - \frac{1}{2^{q+1}}\right) \cdot m.$$

■

Note the importance of the various parameters in the PCP verifier. If we can ensure fewer bits are queried by the verifier, then we prove a stronger hardness bound using the same reduction. Also, we used the simple bound $m \leq 2^q \cdot n^r$, but a sharper bound (i.e. showing that even fewer clauses are in $C(b)$) produces a better hardness lower bound. For example, we can at least make the small observation that $|C(b)| \leq 2^q - 1$, otherwise V does not accept any proof after reading the random bit string b. (in which case we certainly know that $x \notin L$). This leads to a slightly better gap of $1 - 1/(2^{q+1} - 2)$, which is better than $1 - 1/2^{q+1}$ for this constant q. What I'm trying to stress here is that many aspects of how the verifier works are very important to understand the best hardness results we can prove.

Finally, as an interesting side note we can show the easy part of the PCP Theorem: for any constants r, q we have $\mathbf{PCP}_{1,1/2}(r \cdot \log_2 n, q) \subseteq \mathbf{NP}$. Just use the reduction from the proof of Theorem 4 starting with a verifier V for a language $L \in \mathbf{PCP}_{1,1/2}(r \cdot \log_2 n, q)$. This produces a SAT instance that is satisfiable if and only if the original instance is a *yes* instance. Since we have reduced L to a problem in $\mathbf{NP}$ in polynomial time, $L \in \mathbf{NP}$.

30.4 Discussion

The proof of the PCP Theorem as stated in Theorem 3 initially appeared in a paper by Arora, Lund, Motwani, Sudan, and Szegedy [A+98] (it first appeared in 1992). The result is traditionally credited to this work and a slightly earlier work of Arora and Safra [AS98] (again, this first appeared in 1992).

This proof builds on a series of interesting ideas that we do not have time to summarize here. A survey by Ryan O'Donnell gives a great historical overview of the literature leading up to the proof [O05]. Likewise, Håstad's tight hardness bounds for Max-2Lin(3) and Max-3SAT are products of many innovative ideas used to refine the PCP Theorem [H01].

A much simpler proof of the PCP theorem was discovered by Dinur [D07] and can also be found in the book by Arora and Barak [AB09]. This book also contains most of the proof of Håstad's theorems mentioned in these lectures.

References

AB09 S. Arora and B. Barak, *Computational Complexity: A Modern Approach*, Cambridge University Press, 2009.

A+98 S. Arora, C. Lund, R. Motwani, M. Sudan, M. Szegedy, Proof verification and the hardness of approximation, Journal of the ACM, 45(3):501-555, 1998.

AS98 S. Arora and S. Safra, Probabilistic checking of proofs: a new characterization of NP, Journal of the ACM, 45(1):70-122, 1998.

D07 I. Dinur, The PCP theorem by gap amplification, Journal of the ACM, 54(3), 2007.

H01 J. Håstad, Some optimal inapproximability results, Journal of the ACM, 48(4):798–859, 2001.

O05 R. O'Donnell, A history of the PCP theorem, `http://courses.cs.washington.edu/courses/cse533/05au/pcp-history.pdf`