Hardness Of Approximation 2019-04-04

Lecture 1: Introduction

Instructor: Irit Dinur and Amey Bhangale *Scribe: Boaz Menuhin*

In this lecture we introduce approximation algorithms. We will first introduce several Optimization Problems. Afterward, we will present Approximation Algorithms and Gap Problems. We will then cover two variants of the The PCP theorem, and finish with a small example of Property Testing.

1 Optimization Problems

An optimization problem is a problem for which an algorithm should output the "Best" (under some value function) solution. This, as opposed to regular search problems for which one needs to find a solution, no matter how good it is. Here is a list of s few well-known optimization problems.

- `MAX-3-SAT`

  Instance: Given $x_1, ..., x_n$ boolean variables, $C_1, ..., C_m$ clauses, where clause C_i is specified by i_1, i_2, i_3 and by some negation pattern $\in \{0,1\}^3$ i.e. a clause $C_i = x_{i_1} \vee \neg x_{i_2} \vee \neg x_{i_3}$.
  Define:

  - For an assignment $a \in \{0,1\}^n$ $\text{value}(a) \coloneqq \Pr_{1 \le i \le m}[C_i \text{ is satisfied}]$.
  - For an instance ϕ, $\text{value}(\phi) \coloneqq \max_{a \in \{0,1\}^n} \text{value}(a)$.

  Goal: Find $\text{value}(\phi)$.

- `MIN-SET-COVER`

  Instance: A universe V and collection of subsets $S_1, ...S_m \subset V$.
  Goal: find minimal $A \subseteq [m]$ s.t. $\bigcup_{i \in A} S_i = V$.

- `MIN-VERTEX-COVER`

  Given an undirected graph, find the smallest set of vertices that touches all edges.

- `MAX-INDEPENDENT-SET`

  Given a graph, find the largest set of vertices without spanning any edges.

  Remark: Interestingly, despite being very similar in the decision variant, `MIN-VERTEX-COVER` and `MAX-INDEPENDENT-SET` have very different approximation algorithms.

- `MAX-CUT`

  Given a graph, cut it in a way that maximizes the number of edges crossing between it's two parts. i.e. find $S \subset V$ that maximizes
  $$\frac{|E(S, V \setminus S)|}{|E|}.$$

- `MAX-3-LIN`

  Given vars $x_1, ..., x_n$ and equations of the form $x_{i_1} + x_{i_2} + x_{i_3} = b \pmod 2$ with $b \in \{0,1\}$, maximize the number of equations satisfied.

  Fact: Both `MAX-3-LIN` and `MAX-CUT` are NP-hard.

- `MAX-CSP` (Constrains Satisfaction Problem)

  Instance: Given $x_1, ..., x_n$ variables over alphabet Σ (e.g. $\Sigma_{\texttt{SAT}} = \{0,1\}$, $\Sigma_{\texttt{3-COLORING}} = \{1,2,3\}$), $C_1, ..., C_m$ constraints, C_i specified by $i_1, ...i_q \in [n]$ and by some boolean predicate $\varphi : \Sigma^q \to \{0,1\}$
  Define:

  - For an assignment $a \in \Sigma^n$ $\text{value}(a) \coloneqq \Pr_{1 \le i \le m}[C_i \text{ is satisfied}]$.
  - For an instance ψ, $\text{value}(\psi) \coloneqq \max_{a \in \Sigma^n} \text{value}(a)$.

- `MAX-CSP(Φ)`

  This is similar to `MAX-CSP`, except that instead of allowing any predicate, the predicates are form the specified family of predicates $\Phi = \{\varphi_1, ..., \varphi_k\}$.

2 Approximation Algorithms and Gap Problems

All the problems mentioned in the previous section are NP-hard. Thus, if $P \neq NP$, there is no hope of solving it optimally using a polynomial time algorithm. Therefore, we look at finding a solution which is as close to the optimal solution as possible. This gives rise to the following notion of approximate solution.

Definition 2.1 (g-approximation algorithm)**.** *An algorithm A is a g-approximation algorithm for an optimization problem if it runs in polynomial time and it outputs a solution whose value satisfies* $\text{value}(A(x)) \geq g \cdot \text{value}(x)$ *for all instances x.*

Note: This definition applies for maximization problems. For minimization problems we will require $\text{value}(A(x)) \leq g \cdot \text{value}(x)$. In this case, $g \geq 1$.

Let us look at the following two trivial $\frac{1}{2}$-approximation algorithms for `MAX-3-LIN`, one randomized and one deterministic. Given an instance over Boolean variables $x_1, x_2, \ldots, x_n$.

1. **Random Algorithm**: Algorithm sets $x_i \in \{0,1\}$ independently.
2. **Deterministic Algorithm**: Consider the all 0s and all 1s assignments to x_is, one of them should work.

Both satisfy at least $\frac{1}{2}$ of the clauses (first one in expectation), as the $\text{value}(x) \leq 1$ for every instance, an algorithm that satisfies at least half of the clauses is therefore $\frac{1}{2}$-approximation algorithm.

In order to study the complexity of approximation algorithm, it is convenient to look at the following decision version:

Definition 2.2 (Gap Problem)**.** *A `gap`(s,c)`-problem` is the problem of determining for each input x whether x is a*

- ***Yes*** *instance:* $\text{value}(x) \geq c$ *or*
- ***No*** *instance:* $\text{value}(x) \leq s$.

Remark: If $s \leq \text{value}(x) \leq c$ then we don't care and the algorithm can say whatever it wants. This definition suits maximization optimization problems. For minimization optimization problems we will declare x as **Yes** instance if it is below s and as **No** instance if above c.

We now state the celebrated PCP theorem which was proved in early 90s.

Theorem 2.3 (The PCP theorem)**.** *There exists a constant $s < 1$ and a polytime reduction φ taking* `3SAT` *problem to* `gap`$(s,1)$`-CSP` *s.t.*

$$\begin{array}{c} I \in \texttt{3SAT} \\ (i.e.\ \text{value}(I)=1) \end{array} \longrightarrow \text{value}(\varphi(I)) = 1$$
$$\begin{array}{c} I \notin \texttt{3SAT} \\ (i.e.\ \text{value}(I)<1) \end{array} \longrightarrow \text{value}(\varphi(I)) < s$$

An easy corollary of the PCP theorem is that it shows that approximating `MAX-CSP` within a constant factor is NP-hard.

Corollary 2.4. `gap`$(s,1)$`-CSP` *is NP-Hard.*

Following shows the connection between the gap problem and approximation algorithm.

Corollary 2.5. *For every $s' > s$ there is no s'-approximation for* `MAX-CSP` *(assuming $P \neq NP$)*

Proof. Assume toward a contradiction that there exists an algorithm A which achieves s'-approximation. Consider the reduction from Theorem 2.3. If I is a **yes** instance, then $\text{value}(I) = 1 \Rightarrow \text{value}(\varphi(I)) = 1 \Rightarrow \text{value}(A(\varphi(I))) \geq s' \geq s$. If I is a **no** instance, then $\text{value}(\varphi(I)) \leq s$ and $\text{value}(A(\varphi(I))) \leq s$. So deciding if $A(\varphi(I)) > s$ is the same as deciding `gap`$(s,1)$`-CSP`. □

The PCP theorem can be viewed in terms of proof checking paradigm of a language in NP which was the original motivation of proving the theorem. Informally, the PCP Theorem shows that for every language in NP, there exists a polynomial sized proof which can be checked by looking at constantly many locations. More formally,

Theorem 2.6 (Proof system variant ("The original") PCP theorem)**.** *There exists $s < 1$ s.t. every language (problem) in NP has poly-time verifier* `ver` *that*

- *reads input x ($|x| = n$),*
- *tosses $O(\log(n))$ coins,*
- *and reads $O(1)$ bits from proof π*

s.t.

$$\begin{array}{c} x \in L \\ (e.g.\ I \in \texttt{3SAT}) \end{array} \longrightarrow \exists \pi\ Pr\left[\texttt{ver}^{\pi}(x)\ \ accpets\right] = 1$$
$$\begin{array}{c} x \notin L \\ (e.g.\ I \notin \texttt{3SAT}) \end{array} \longrightarrow \forall \pi\ Pr\left[\texttt{ver}^{\pi}(x)\ \ accpets\right] < s$$

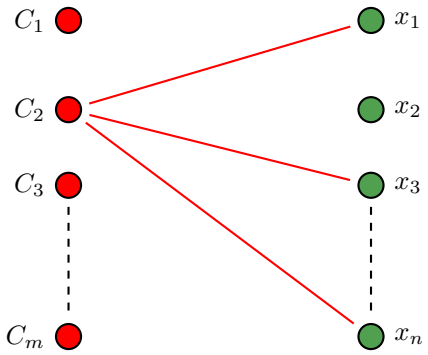
Question: So what is s?
Answer: Håstad [**?**] found the value of s for `3SAT` ($s = \frac{7}{8} + \epsilon$) by showing `gap`$(\frac{7}{8}+\epsilon,1)$`-3SAT` is NP-hard for all $\epsilon > 0$.

Question: What is the smallest value of s for which hardness holds? What is the smallest number of queries?
Answer: Depends on the alphabet and we will look more into in in the upcoming lectures.

We now define a famous problem in the area of hardness of approximation called Label Cover.

Definition 2.7 (`LABEL-COVER`)**.** ***Instance****: A bipartite graph $G = (R, L, E)$. For every vertex v there is a set of labels Σ_v. For every edge (u, v) there is a set of allowed labels pairs $\Pi_{u,v} \subseteq \Sigma_u \times \Sigma_v$.*
Goal*: Find an assignment σ s.t. for every v, $\sigma(v) \in \Sigma_v$ and maximize $Pr\left[(\sigma_u, \sigma_v)\ \in \Pi_{u,v}\right]$*

Figure 1: `LABEL-COVER` instance made from `gap`$(s,1)$`-CSP` instance



(a) variables x_1, x_3, x_n participate in constraint C_2.

Theorem 2.8 (Baby theorem)**.** *There exists* $s_1 < 1$ *such that gap*$(s_1, 1)$`-LABEL-COVER` *is NP-Hard.*

Proof. Starting from `gap`$(s,1)$`-CSP` instance with $x_1, ..., x_n$ vars and $C_1, ..., C_m$ constraints, we construct a `LABEL-COVER` instance $G = (R, L, E)$ (as in Figure 1). R consists of a vertex for each x_i and Σ_{x_i} is the Σ of the `CSP` (e.g. in `3SAT` $\Sigma_{x_i} = \{0,1\}$). L consists of a vertex for each C_j and Σ_{C_j} is a set of all satisfying assignments for C_j. $(C_j, x_i) \in E$ if C_j contains x_i (x_i participates in C_j). And finally,

$\Pi_{C_j,x_i} = \{(\alpha,\beta) | \alpha$ is an assignment for the variables of C_j, which *satisfies* C_j **and** agrees with β on $x_i\}$.

One can show that the reduction works! In other words, satisfied instances are mapped to Label Cover instance with value 1 and if the value of original instance is at most $s < 1$, then the value of the Label Cover instance is bounded away from 1. □

Notice: the `LABEL-COVER` instance have a **Projection Property** - *any* assignment to c_i forces a unique assignment for x_j. We will see more of this property later in the course.

3 Property Testing

Property testing algorithm is an algorithm which decides whether an input satisfies a property or that it is "far" from it. Similar to what done in PCP, in property testing, the amount of bits read from the input is sublinear (or even constant), however the proof is usually the input itself. Historically, property testing originates from PCP.

Codes

How did Håstad prove his theorem on `gap`$(\frac{7}{8}+\epsilon, 1)$`-3SAT` being NP-hard? In the next lecture, we will show that `gap`$(s,1)$`-LABEL-COVER` is NP-hard for *any* constant $s > 0$. Håstad used this as a starting point and used some very redundant encoding with local testability property to prove hardness result for `MAX-3-SAT`.

Start: with a `LABEL-COVER` instance

Idea: Use locally testable codes, encode labels of the left and right vertices by a function $f : \{0,1\}^t \to \{0,1\}$. For our purpose, a code is a set of strings far apart from each other.

Hadamard Code

Hadamard code is a set of all linear functions $f(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i \cdot a_i \mod 2$

$$\texttt{HAD} = \{\{0,1\}^{2^n} | \exists a_1, ..., a_n \text{ s.t. } f(x_1, ..., x_n) = \sum_{i=1}^{n} a_i \cdot x_i\}$$

Fact: The size of the set is 2^n.

Locally Testable Code

There is a tester which is an algorithm that randomly selects $i_1, ..., i_q$, reads the codeword in these locations and accepts/rejects.

- $f \in$ code $\Rightarrow Pr$ [tester accepts] $= 1$
- f far from code $\Rightarrow Pr$ [tester accepts] $< s$.

The following tester for $f : \{0,1\}^n \to \{0,1\}$ is a good tester for linearity.

Theorem 3.1 (Linearity Testing Theorem [**?**])**.** *First, choose $x, y \in \{0,1\}^n$ uniformly at random. Then, read $f(x), f(y), f(x+y)$ and accept iff $f(x) + f(y) = f(x+y)$.*

- *if f is linear ($\exists a_1, ..., a_n$ s.t. $f(x_1, ..., x_n) = \sum a_i \cdot x_i$) then Pr [test accepts] $= 1$.*
- *If there is no linear function that agrees with f on $\frac{1}{2}+\epsilon$ fraction of the inputs, then Pr [test accepts] $< \frac{1}{2} + \epsilon$.*

In the next lecture, we will study the tools to prove the above theorem.