CSC2429: Proof Complexity **Winter 2018**

Lecture 1: Introductions

Lecturer: Toni Pitassi *Scribe: Lily Li*

1.1 Administration

Similar courses include: Boak Barak and Steurer/Kothari. Notes for the the Barak and Steurer lectures that tie into these notes will be adding in blue. Notes from the Massico Lauria lectures will be added in red. Choice between two assignments or an assignment and one presentation.

1.2 Introductions Proper

Definition 1.1 *A* ***proof system*** *is a* non-deterministic *procedure for generating a family of algorithms. These can be used to* rule out *approaches for solving problem using certain systems e.g. SA or SOS.*

What is an *extended formulation*? This is a large class of linear programs. There is a clever reduction that reduces this to SA (thus SA results are lower bounds for the LPs).

The point of a proof complexity *upper bound* is to produce efficient algorithms.

1.2.1 Motivating Examples

Example 1.2 MaxSAT*: given $f = c_1 \wedge \cdots \wedge c_m$ which is a 3CNF formula over the variables $x_1, ..., x_n$. Assign $x_i \in \{0, 1\}$ to maximize the number of satisfied clauses.*

Consider the integer program formulation: let the variables be $x_1, ..., x_n$ and $c_1, ..., c_m$. We want $\max \sum c_i$ where c_i is set to 1 if it is satisfied. Each clause gets turned into a formula in the usual way. If $c_1 = x_1 \wedge \neg x_2 \wedge x_3$ then c_1 becomes the system

$$x_1 + (1 - x_2) + x_3 \geq c_1$$

where $x_i, c_j \in \{0, 1\}$. The standard approach is to relax the LP to an IP and solve the LP (of course you add the constraints $x_i, c_j \in [0, 1]$).

How does the $FRACTOPT$ compare to the integral OPT? If f is an exact 3CNF and f is unsatisfiable then $OPT \geq \frac{7}{8}m$, where m is the number of clauses. To see this remark that if every triple of terms appeared in the formula f then one out of every eight clauses must be false; general formulas might have fewer clauses and thus fewer false clauses. Unfortunately, if we run the fraction algorithm, $FRACTOPT = m$ by choosing each $x_i = \frac{1}{2}$ (this allows each $c_i = 1$). Generally

$$FRACTOPT \geq OPT \geq \mathrm{rd}(FRACTOPT)$$

and we compare the quality of our rounded by

$$\frac{\mathrm{rd}(FRACTOPT)}{OPT} = \frac{\mathrm{rd}(FRACTOPT)}{FRACTOPT} \cdot \frac{FRACTOPT}{OPT} \geq K$$

for some constant K and usually this is done by showing that $\frac{\mathrm{rd}(FRACTOPT)}{FRACTOPT} \geq K$. Given a integer program and its relaxation, the ratio

$$\frac{OPT}{FRAC}$$

is the **integrality gap** of the relaxation.

Example 1.3 MaxCUT*: given a weighted graph $G = (V, E)$ with weight function w find a cut S which induces the largest edge set.*

The natural formulation here is a quadratic program: let the variables be y_i for every vertex $v_1, ..., v_n$ then

$$\max \frac{1}{2} \sum_{i<j} w_{i,j}(1 - y_i y_j)$$

where $y_i \in \{-1, 1\}$ for $i \in [n]$. Let the optimum value of this program be $\mathsf{opt}(G)$*. We attempt to write down a semidefinite program whose value is an* upper bound *on* $\mathsf{opt}(G)$*.*

Here comes that crazy semi-definite relaxation: replace each y_i by a vector $\mathbf{u}_i \in S^{n-1} = \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}\| = 1\}$ — read: the unit sphere in $n-1$ dimensions. Our semi-definite relaxation becomes:

$$\text{Maximize} \quad \frac{1}{2} \sum_{i<j} w_{i,j}(1 - \mathbf{u}_i^T \mathbf{u}_j) \tag{1.1}$$

$$\text{Subject to} \quad \mathbf{u}_i \in S^{n-1}, \quad i \in [n] \tag{1.2}$$

Note to convert y_i into a vector $\mathbf{u}_i$ it suffices to write $\mathbf{u}_i = (0, ..., 0, y_i)$. Further observe that every solution to the original optimization problem is a solution to our relaxation (the relaxation could contain other solutions as well). We need one more substitution to get to our actual semi-definite program, namely $x_{i,j} = \mathbf{u}_i^T \mathbf{u}_j$. The system now becomes

$$\text{Maximize} \quad \frac{1}{2} \sum_{i<j} w_{i,j}(1 - x_{i,j}) \tag{1.3}$$

$$\text{Subject to} \quad x_{i,i} = 1 \quad i \in [n] \text{ and } X \succeq 0. \tag{1.4}$$

This works because X can be written as $X = U^T U$ where $\mathbf{u}_i$ is the i^{th} column of U. Thus every solution to 1.1 is a solution of 1.3 and vice versa (there is some nuance here regarding the constraints). Then the optimum values of the semidefinite program $\mathsf{SDP}(G) \geq \mathsf{opt}(G)$ *and we can find the optimum matrix $X^* \succeq 0$ with $x^*_{i,i} = 1$ for all $i \in [n]$ satisfying*

$$\sum_{i<j} w_{i,j}(1 - x^*_{i,j}) \geq \mathsf{SDP}(G) - \epsilon$$

for every $\epsilon > 0$ in polynomial time (don't ask how... I don't know yet). We find the Cholesky factorization *of X^* into $(U^*)^T U^*$ and take the columns of U^* to be an almost-optimal solution to the vector problem.*

Finally we need to round the vector solutions to get a feasible integer valued solution. Do this by exploiting the geometry of the $\mathbf{u}_i$'s. Since $\mathbf{u}_i \in S^{n-1}$ we will randomly *pick a vector $\mathbf{p} \in S^{n-1}$ and use the normal to $\mathbf{p}$ to divide the plane into two halves. If $\mathbf{u}_i$ is in one half then round $\mathbf{u}_i$ to 1 otherwise round $\mathbf{u}_i$ to -1.*

The reason why this works relies on a somewhat cryptic probability argument. After taking an appropriate derivative you find that the expected number of cut edges is at least $0.878\mathsf{opt}(G)$*.*

An alternative formulation of the max-cut problem for an n-vertex graph G is as the following degree two polynomial $f_G(x)$ where $\mathbf{x}$ is a cut:

$$f_G(x) = \sum_{e_{i,j} \in E} (x_i - x_j)^2.$$

By choosing $x_i \in \{0,1\}$ — thus choosing $\mathbf{x} \in \{0,1\}^n$ — we effectively get to choose the partition. Deciding if $c - f_G(\mathbf{x}) \geq 0$ for all $\mathbf{x}$ is the same as deciding if the maximum cut of G is greater than c.

The above is a special cases of the problem we want to consider. These *non-negativity over the hyper-cube* problems ask: Given a low-degree polynomial $f : \{0,1\}^n \to \mathbb{R}$, decided if $f \geq 0$ over the hyper-cube or if there exists a point $\mathbf{x} \in \{0,1\}^n$ such that $f(x) < 0$.

Definition 1.4 *The* ***sum-of-squares algorithm****, when restricted to the above special cases, takes as input the polynomial f and outputs either (1) a proof that $f(\mathbf{x}) \geq 0$ for all $\mathbf{x}$ or (2) a collection of objects which represent points $\mathbf{x}$ such that $f(\mathbf{x}) < 0$.*

An appropriate certificate is the following:

Definition 1.5 *A degree-d* ***sum-of-squares certificate*** *(of non-negativity) for $f : \{0,1\}^n \to \mathbb{R}$ is a set of polynomials $g_1, ..., g_r : \{0,1\}^n \to \mathbb{R}$ of degree $\leq \frac{d}{2}$ for some $r \in \mathbb{N}$ such that*

$$f(\mathbf{x}) = g_1^2(\mathbf{x}) + \cdots + g_r^2(\mathbf{x})$$

for every $\mathbf{x} \in \{0,1\}^n$. This set of polynomials is a degree-d sos proof *of the inequality $f \geq 0$.*

1.2.2 Resolution

Think of this as a system which *tightens* linear programs. But first: proof system basics!

Definition 1.6 *A* ***proof system*** *takes as input a set of constraints (CSP) over $x_1, ..., x_n$ usually $x_i \in \{0,1\}$ or $\{-1,1\}$ (sometimes $x_i \in \mathbb{F}, \mathbb{R}_{\geq 0}$) and then*

$$\mathcal{P}(y) \to F$$

that is the proof system takes as a proof (string) y and outputs what y is a proof of, namely F (generally we think of these are unsatisfiable CSPs). We want our algorithm to run in polynomial time and the proof to be complete and sound.

Definition 1.7 *A proof system $\mathcal{P}$ is* ***polynomially-automatizable*** *if there for all n sufficiently large there exists an algorithm A such that $A(F)$ — F is $KCNF$ in n variables — outputs a valid $\mathcal{P}$-proof (that is: $\mathcal{P}(A(F)) = F$) and such that the running time of A is polynomial in the size of the shortest $\mathcal{P}$-proof of F.*

1.2.2.1 Basics

Resolution is a refutation based proof system which operates on CNF (with variables which take values $\{0,1\}$). You take clauses and resolve new clauses. If you ever get the empty clause then the initial CNF formula is unsatisfiable. You could have a decision tree or DAG versions, but these are actually truth tables which quit early.

The way you want to think about this is as follows: replace each $\{0,1\}$ valued x_i with fractional $0 \leq x_i \leq 1$. The constraints are extended when we derive new clauses, and this shrinks the size of the poly-tope containing all the feasible solutions.

In-terms of automizability, a tree-resolution proofs can be found in time $n^{\log(\text{size of the shortest proof})}$ where size is the size of the resolution tree. There is a BenSasson-Wigderson theorem that helps you show the above.

Example 1.8 *Consider a 3CNF formula over $x_1, ..., x_n$ and you want to decide if F is satisfiable or not. Surprisingly the above resolution technique bets the brute-force approach.*