9.4 Lift and Project Methods

In both examples we saw thus far, the integrality gap proved was tight. Can we design tighter relaxations for these problems? Researchers have looked at this question in great detail. Next, we consider an more abstract view of the process of writing better relaxation.

The feasible region for the ILP problem is a polytope, namely, the convex hull of the *integer* solutions [2]. We will call this the *integer polytope*. The set of feasible solutions of the relaxation is also a polytope, which contains the integer polytope. We call this the *relaxed polytope*; in the above examples it was of polynomial size but note that it would suffice (thanks to the Ellipsoid algorithm) to just have an implicit description of it using a polynomial-time separation oracle. On the other hand, if **P** $\neq$ **NP** the integer polytope has no such description. The name of the game here is to design a relaxed polytope that as close to the integer polytope as possible. *Lift-and-project* methods give, starting from any relaxation of our choice, a hierarchy of relaxations where the final relaxation gives the integer polytope. Of course, solving this final relaxation takes exponential time. In-between relaxations may take somewhere between polynomial and exponential time to solve, and it is an open question in most cases to determine their integrality gap.

Basic idea

The main idea in the *Lift and Project* methods is to try to simulate non-linear programming using linear programming. Recall that nonlinear constraints are very powerful: to restrict a variable x to be in $\{0,1\}$ we simply add the quadratic constraint $x(1-x)=0$. Of course, this means nonlinear programming is NP-hard in general. In lift-and-project methods we introduce auxiliary variables for the nonlinear terms.

EXAMPLE 10 Here is a quadratic program for the vertex cover problem.

$0 \leq x_i \leq 1$ for each $i \in V$

$(1-x_i)(1-x_j)=0$ for each $\{i,j\} \in E$

To *simulate* this using an LP, we introduce extra variables $y_{ij} \geq 0$, with the intention that y_{ij} represents the product $x_i x_j$. This is the **lift** step, in which we *lift* the problem to a higher dimensional space. To get a solution for the original problem from a solution for the new problem, we simply **project** it onto the variables x_i. Note that this a relaxation of the original integer linear program, in the sense that any solution of that program will still be retained as a solution after the lift and project steps. Since we have no way of ensuring $y_{ij} = x_i x_j$ in every solution of the lifted problem, we still may end up with a relaxed polytope. But note that this relaxation can be no worse than the original LP relaxation (in which we simply dropped the integrality constraints), because $1 - x_i - x_j + y_{ij} = 0$, $y_{ij} \geq 0 \Rightarrow x_i + x_j \geq 1$, and any point in the new relaxation is present in the original one.

(If we insisted that the matrix (y_{ij}) formed a positive semi-definite matrix, it would still be a (tighter) relaxation, and we get a Semi-definite Programming problem. We shall see this in the next lecture.)

9.5 Sherali-Adams Lift and Project Method

Now we describe the method formally. Suppose we are given a polytope $P \subset \mathbf{R}^n$ (via a separation oracle) and we are trying to get a representation for $P_0 \subset P$ defined as the convex hull of $P \cap \{0,1\}^n$. We proceed as follows:

[2] By integer solutions, we refer to solutions with co-ordinates 0 or 1. We assume that the integrality constraints in the ILP correspond to restricting the solution to such points.

The first step is *homogenization*. We change the polytope P into a *cone* K in $\mathbf{R}^{n+1}$. (A *cone* is a set of points that is closed under scaling: if x is in the set, so is $c \cdot x$ for all $c \geq 0$.) If a point $(\alpha_1, \ldots, \alpha_n) \in P$ then $(1, \alpha_1, \ldots, \alpha_n) \in K$. In terms of the linear constraints defining P this amounts to multiplying the constant term in the constraints by a new variable x_0 and thus making it homogeneous: i.e., $\sum_{i=1}^{n} a_i x_i \geq b$ is replaced by $\sum_{i=1}^{n} a_i x_i \geq b x_0$. Let $K_0 \subset K$ be the cone generated by the points $K \cap \{0,1\}^{n+1}$ ($x_0 = 0$ gives the origin; otherwise $x_0 = 1$ and we have the points which define the polytope P_0).

For $r = 1, 2, \ldots, n$ we shall define $SA^r(K)$ to be a cone in $\mathbf{R}^{V_{n+1}(r)}$, where $V_{n+1}(r) = \sum_{i=0}^{r} \binom{n+1}{i}$. Each co-ordinate corresponds to a variable y_s for $s \subset [n+1]$, $|s| \leq r$. The intention is that the variable y_s stands for the homogenuous term $(\prod_{i \in s} x_i) \times x_0^{r-|s|}$. Let $\mathbf{y^{(r)}}$ denote the vector of all the $V_{n+1}(r)$ variables.

DEFINITION 28 *Cone $SA^r(K)$ is defined as follows:*

- *$SA^1(K) = K$, with $y_{\{i\}} = x_i$ and $y_\phi = x_0$.*
- *$SA^r(K)$ The constraints defining $SA^r(K)$ are obtained from the constraints defining $SA^{r-1}(K)$: for each constraint $\mathbf{ay^{(r-1)}} \geq 0$, for each $i \in [n]$, form the following two constraints:*

$$(1 - x_i) * \mathbf{ay^{(r-1)}} \geq 0$$

$$x_i * \mathbf{ay^{(r-1)}} \geq 0$$

where the operator "$*$" distributes over the sum $\mathbf{ay^{(r-1)}} = \sum_{s \subset [n]: |s| \leq r} a_s y_s$ and $x_i * y_s$ is a shorthand for $y_{s \cup \{i\}}$.

Suppose $(1, x_1, \ldots, x_n) \in K \cap \{0,1\}^{n+1}$. Then we note that the cone $SA^r(K)$ contains the points defined by $y_s = \prod_{i \in s} x_i$. This is true for $r = 1$ and is maintained inductively by the constraints we form for each r. Note that if $i \in s$ then $x_i * y_s = y_s$, but we also have $x_i^2 = x_i$ for $x_i \in \{0,1\}$.

To get a relaxation of K we need to come back to $n+1$ dimensions. Next we do this:

DEFINITION 29 *$S^r(K)$ is the cone obtained by projecting the cone $SA^r(K)$ to $n+1$ dimensions as follows: a point $\mathbf{u} \in SA^r(K)$ will be projected to $\mathbf{u}|_{s:|s| \leq 1}$; the variable $\mathbf{u}_\phi$ is mapped to x_0 and for each $i \in [n]$ $\mathbf{u}_{\{i\}}$ to x_i.*

EXAMPLE 11 This example illustrates the above procedure for Vertex Cover, and shows how it can be thought of as a "simulation" of non-linear programming. The constraints for $S^1(K)$ come from the linear programming constraints:

$$\forall j \in V, 0 \leq y_{\{j\}} \leq y_\phi$$

$$\forall \{i,j\} \in E, y_{\{i\}} + y_{\{j\}} - y_\phi \geq 0$$

Among the various constraints for $SA^2(K)$ formed from the above constraints for $SA^1(K)$, we have $(1 - x_i) * (y_\phi - y_{\{j\}}) \geq 0$ and $(1 - x_i) * (y_{\{i\}} + y_{\{j\}} - y_\phi) \geq 0$ The first one expands to $y_\phi - y_{\{i\}} - y_{\{j\}} + y_{\{i,j\}} \geq 0$ and the second one becomes to $y_{\{j\}} - y_{\{i,j\}} - y_\phi - y_{\{i\}} \geq 0$, together enforcing $y_\phi - y_{\{i\}} - y_{\{j\}} + y_{\{i,j\}} = 0$. This is "simulating" the quadratic constraint $1 - x_i - x_j + x_i x_j = 0$ or the more familiar $(1 - x_i)(1 - x_j) = 0$ for each $\{i,j\} \in E$. It can be shown that the defining constraints for the cone $S^2(K)$ are the odd-cyle constraints: "for an odd cycle C, $\sum_{x_i \in C} \geq (|C| + 1)/2$." An exact characterization of $S^(K)r)$ for $r > 2$ is open.

Intuitively, as we increase r we get tighter relaxations of K_0, as we are adding more and more "valid" constraints. Let us prove this formally. First, we note the following characterization of $SA^r(K)$.

LEMMA 49
$\mathbf{u} \in SA^r(K)$ *iff* $\forall i \in [n]$ *we have* $\mathbf{v}^i, \mathbf{w}^i \in SA^{r-1}(K)$, *where* $\forall s \subset [n]$, $|s| \leq r-1$, $\mathbf{v}^i = \mathbf{u}_{s\cup\{i\}}$ *and* $\mathbf{w}^i = \mathbf{u}_s - \mathbf{u}_{s\cup\{i\}}$.

PROOF: To establish the lemma, we make our notation used in defining $SA^r(K)$ from $SA^{r-1}(K)$ explicit. Suppose $SA^{r-1}(K)$ has a constraint $\mathbf{a}\mathbf{y}^{(\mathbf{r-1})} \geq 0$. Recall that from this, for each $i \in [n]$ we form two constraints for $SA^r(K)$, say $\mathbf{a}'\mathbf{y}^{(\mathbf{r})} \geq 0$ and $\mathbf{a}''\mathbf{y}^{(\mathbf{r})} \geq 0$, where $\mathbf{a}'\mathbf{y}^{(\mathbf{r})} \equiv x_i * \mathbf{a}\mathbf{y}^{(\mathbf{r-1})}$ is given by

$$\mathbf{a}'_s = \begin{cases} 0 & \text{if } i \notin s \\ \mathbf{a}_s + \mathbf{a}_{s\setminus\{i\}} & \text{if } i \in s \end{cases} \tag{9.5.1}$$

and $\mathbf{a}''\mathbf{y}^{(\mathbf{r})} \equiv (1 - x_i) * \mathbf{a}\mathbf{y}^{(\mathbf{r-1})}$ by

$$\mathbf{a}''_s = \begin{cases} \mathbf{a}_s & \text{if } i \notin s \\ -\mathbf{a}_{s\setminus\{i\}} & \text{if } i \in s \end{cases} \tag{9.5.2}$$

Then we see that

$$\begin{aligned} \mathbf{a}'\mathbf{u} &= \sum_{s\ni i} (\mathbf{a}_s + \mathbf{a}_{s\setminus\{i\}})\, \mathbf{u}_s \\ &= \sum_{s\ni i} \mathbf{a}_s \mathbf{v}^i_s + \sum_{s\not\ni i} \mathbf{a}_s \mathbf{v}^i_s = \mathbf{a}\mathbf{v}^i \end{aligned}$$

where we used the fact that for $s \ni i$, $\mathbf{u}_s = \mathbf{v}^i_s = \mathbf{v}^i_{s\setminus\{i\}}$. Similarly, noting that for $s \ni i$, $\mathbf{w}^i_s = 0$ and for $s \not\ni i$, $\mathbf{w}^i_s = \mathbf{u}_s - \mathbf{u}_{s\setminus\{i\}}$, we have

$$\begin{aligned} \mathbf{a}''\mathbf{u} &= \sum_{s\ni i} -\mathbf{a}_{s\setminus\{i\}}\mathbf{u}_s + \sum_{s\not\ni i} \mathbf{a}_s \mathbf{u}_s \\ &= \sum_{s\not\ni i} (-\mathbf{a}_s \mathbf{u}_{s\cup\{i\}} + \mathbf{a}_s \mathbf{u}_s) = \mathbf{a}\mathbf{w}^i \end{aligned}$$

Therefore, $\mathbf{u}$ satisfies the constraints of $SA^r(K)$ iff for each $i \in [n]$ $\mathbf{v}^i$ and $\mathbf{w}^i$ satisfy the constraints of $SA^{r-1}(K)$. □

Now we are ready to show that $S^r(K)$, $1 \leq r \leq n$ form a hierarchy of relaxations of K_0.

THEOREM 50
$K_0 \subset S^r(K) \subset S^{r-1}(K) \subset K$ *for every* r.

PROOF: We have already seen that each integer solution $\overline{x}$ in K_0 gives rise to a corresponding point in $SA^r(K)$: we just take $\mathbf{y}^{()}s = \prod_{i\in s} x_i$. Projecting to a point in $S^r(K)$, we just get $\overline{x}$ back. Thus $K_0 \subset S^r(K)$ for each r.

So it is left to only show that $S^r(K) \subset S^{r-1}(K)$, because $S^1(K) = K$.

Suppose $\mathbf{x} \in S^r(K)$ is obtained as $\mathbf{u}|_{s:|s|\leq 1}$, $\mathbf{u} \in SA^r(K)$. Let $\mathbf{v}^i, \mathbf{w}^i \in SA^{r-1}(K)$ be two vectors (for some $i \in [n]$) as given by Lemma 49. Since $SA^{r-1}(K)$ is a convex cone, $\mathbf{v}^i + \mathbf{w}^i \in SA^{r-1}(K)$. Note that for each $s \subset [n]$, $|s| \leq r-1$, $\mathbf{w}^i_s + \mathbf{v}^i_s = \mathbf{u}_s$. In particular this holds for s, $|s| \leq 1$. So $\mathbf{x} = (\mathbf{v}^i + \mathbf{w}^i)|_{s:|s|\leq 1} \in S^{r-1}(K)$. □

THEOREM 51
$S^n(K) = K_0$

PROOF: Recall from the proof of Theorem 50 that if $\mathbf{x} \in S^r(K)$, then there are two points $\mathbf{v}^i, \mathbf{w}^i \in SA^{r-1}(K)$ such that $\mathbf{x} = \mathbf{v}^i|_{s:|s|\leq 1} + \mathbf{w}^i|_{s:|s|\leq 1}$. It follows from the definition of $\mathbf{v}^i$ and $\mathbf{w}^i$ that $\mathbf{v}^i_\phi = \mathbf{v}^i_{\{i\}}$ and $\mathbf{w}^i_{\{i\}} = 0$. So $\mathbf{v}^i|_{s:|s|\leq 1} \in S^{r-1}(K)|_{y_{\{i\}}=y_\phi}$ and $\mathbf{w}^i|_{s:|s|\leq 1} \in S^{r-1}(K)|_{y_{\{i\}}=0}$. Hence, $S^r(K) \subset S^{r-1}(K)|_{y_{\{i\}}=y_\phi} + S^{r-1}(K)|_{y_{\{i\}}=0}$. Further, this holds for all $i \in [n]$. Thus,

$$S^r(K) \subset \bigcap_{i\in[n]} \left(S^{r-1}(K)|_{y_{\{i\}}=y_\phi} + S^{r-1}(K)|_{y_{\{i\}}=0}\right)$$

Repeating this for $r-1$ and so on, we get

$$S^r(K) \subset \bigcap_{\{i_1,\ldots,i_r\}\subset[n]} \left(\sum_{T\in\{0,y_\phi\}^r} K|_{(y_{\{i_1\}},\ldots,y_{\{i_r\}})=T} \right)$$

For $r = n$ this becomes simply

$$S^n(K) \subset \sum_{T\in\{0,y_\phi\}^n} K|_{(y_{\{1\}},\ldots,y_{\{n\}})=T} = K_0$$

Along with $K_0 \subset S^n(K)$ this gives us that $S^n(K) = K_0$. □

Finally we note that for small r we can solve an optimization problem over $S^r(K)$ relatively efficiently.

THEOREM 52
If there is a polynomial (in n) time separation oracle for K, then there is an $n^{O(r)}$ algorithm for optimizing a linear function over $S^r(K)$.

PROOF: Note that using Lemma 49, a separation oracle for $S^r(K)$ can be formed by calling the separation oracle for $S^{r-1}(K)$ $2n$ times (for $\mathbf{v}^i$ and $\mathbf{w}^i$, $i \in [n]$), and therefore, calling the separation oracle for K $n^{O(r)}$ times. Given access to the separation oracle, the optimization can be carried out using the ellipsoid method in polynomial time. □

Thus, on the one hand a higher value of r gives a tighter relaxation (a smaller integrality gap) and possibly a better approximation (with $r = n$ giving the exact solution), but on the other hand the time taken is exponential in r. So to use this method, it is crucial to understand this trade-off. It has been shown by Arora, Bollobas and Lovasz that in applying a related method by Lovasz and Schriver to the Vertex cover problem, even for $r = \Omega(\sqrt{\log n})$ the integrality gap is as bad as $2 - o(1)$. Sanjeev conjectures that for somewhat higher r the integrality gap may go down significantly. But this problem is open.