CSC2411 - Linear Programming and Combinatorial Optimization*
Lecture 12: The Lift and Project Method

Notes taken by Stefan Mathe

April 28, 2007

Summary: Throughout the course, we have seen the importance of obtaining relaxations of integer problems that are as tight as possible in terms of their integrality gap. The "lift and project" method is introduced, which builds such relaxation in a systematic fashion. The method is illustrated on the familiar Vertex Cover problem. Some complexity results from the literature are summarized in the end.

1 Motivation

The second half of the course has concentrated on relaxations and using them to come up with approximation algorithms. The answer one gets may depend on the relaxation used, and one can attempt to strengthen the relaxation to get a better approximation.

Let us look at a particular instance of this process. Recall the relaxation of the Vertex Cover Problem problem:

$$\min \sum_{i \in V} x_i$$

s.t.

$$x_i + x_j \geq 1,\ i,j \in E(G)$$
$$0 \leq x_i \leq 1$$

In lecture 10, section 3, we have shown that this problem has an integrality gap of 2 (the complete graph is one instance in which this worst case integrality gap is attained).

A natural question to ask at this point is whether there is a way to tighten this relaxation up so that the integrality gap is reduced. In the following section, we shall revisit one such attempt to relaxation.

* Lecture Notes for a course given by Avner Magen, Dept. of Computer Sciecne, University of Toronto.

2 Odd-cycle constraints

One candidate tightening we might try has been explored in Assignment 3, namely adding odd-cycle constraints.

The idea is to add one constraint for each odd cycle in the graph, which imposes that the sum of the values assigned to all the vertices in an arbitrary cycle C be lower bounded by half of the length of the cycle plus one. Formally, the linear problem becomes:

$$\min \sum_{i \in V(G)} x_i$$

s.t.

$$\begin{aligned} x_i + x_j \geq 1 & \quad \forall i, j \in E(G) \quad \textit{(edge constraints)} \\ \sum_{i \in C} x_i \geq \frac{|C|+1}{2} & \quad \forall C \text{ odd cycle in G} \quad \textit{(odd cycle constraints)} \\ x_i \geq 0 & \quad \forall i \in E(G) \quad \textit{(non-negativity constraints)} \end{aligned}$$

To see the motivation behind adding these constraints, let us consider the case of a triangle. Clearly, in the integral case, we have to select at least two of the vertices to cover all the edges. Consider just having the edge constraints for the triangle:

$$\begin{aligned} x_1 + x_2 &\geq 1 \\ x_2 + x_3 &\geq 1 \\ x_1 + x_3 &\geq 1 \end{aligned}$$

We can try to derive a constraint for the entire triangle by adding the inequalities above and diving by 2, which leads to:

$$x_1 + x_2 + x_3 \geq 1.5$$

This is clearly a weaker constraint than the odd-cycle constraint for the triangle, which is:

$$x_1 + x_2 + x_3 \geq 2$$

Hence the odd cycle constraint is a strengthening of the relaxation that holds in the integral case.

Alas, it can be shown that the integrality gap with odd-cycle constraints is still 2.

3 Lift and project for Vertex Cover

The choice for adding odd-cycle constraints was ad-hoc, and there are many other ways to relax the problem (in fact, the number of possible constraints we could add is exponential - this follows from the NP-Completeness of vertex cover, see [Tou06], page 12).

In what follows, we shall investigate a systematic way of building relaxations to the Vertex Cover problem. In this section, we shall begin with a relaxation of Vertex Cover as an illustrative example for the method. In the next section, we shall formally define the method in the general setting.

3.1 Vertex Cover as a Quadratic Program

The starting point is the observation that we can write Vertex Cover as a quadratic program. Remember that Vertex Cover is an integer program. Hence we must start by imposing the constraint that $x_i \in \{0, 1\}$. This can easily be achieved by adding the following quadratic constraints:

$$x_i(1 - x_i) = 0, \ \forall i \in V(G)$$

Clearly, only values in the set $\{0, 1\}$ satisfy the above constraint.

We can also transform edge constraints into quadratic inequalities using a similar strategy:

$$(1 - x_i)(1 - x_j) = 0, \ \forall i, j \in E(G)$$

This leads us to the following quadratic program (after performing the multiplications):

$$\min \sum_{i \in V(G)} x_i$$

s.t.

$$\begin{aligned} x_i - x_i^2 &= 0 \qquad && \forall i \in V(G) \\ 1 - x_i - x_j + x_i x_j &= 0 \qquad && \forall i, j \in E(G) \end{aligned}$$

Since this quadratic program is an equivalent description of Vertex Cover, we know that we cannot hope to solve it in polynomial time.

3.2 Lifting the dimensionality

The next step is to relax this quadratic program to get a linear program. This can be achieved by replacing each term of the form $x_i x_j$ with a new nonnegative variable y_{ij}. The resulting linear problem is:

$$\min \sum_{i \in V(G)} x_i$$

s.t.

$$\begin{aligned} x_i - y_{ii} &= 0 \qquad &\forall i \in V(G) \\ 1 - x_i - x_j + y_{ij} &= 0 \qquad &\forall i,j \in E(G) \\ x_i &\geq 0 \qquad &\forall i \in V(G) \\ y_{ij} &\geq \qquad &\forall i,j \in E(G) \end{aligned}$$

This is a linear relaxation problem that has $n + n^2$ variables. One question to ask is whether this relaxation is stronger or weaker than the straightforward relaxation of Vertex Cover? It easy to see that for each edge i, j we have:

$$x_i + x_j = 1 + y_{ij} \geq 1$$

Hence by first rewriting vertex cover as a quadratic program and the performing relaxation in the manner presented above we have gained something, namely that we have obtained a stronger relaxation.

Can we impose more constraints on the y_{ij} variables? One immediate method that comes to mind is:

$$y_{ij} = y_{ji} \ , \forall i, j \in V(G)$$

Notice that this constraint does not influence the solution in the case of Vertex Cover (because for each pair $i, j \in V(G)$, at most one of the variables y_{ij} and y_{ji} appears in the constraints), but there are other combinatorial problems where adding these constraints further tightens the resulting relaxation.

3.3 Semidefinie programming variant

We can tighten the relaxation we presented before even further, by imposing that the matrix $\mathbf{Y} = (y_{ij})$ formed by the variables y_{ij} be PSD. Of course, this will move us out of the linear programming domain into the semidefinite programming realm:

$$\min \sum_{i \in V(G)} y_{ii}$$

s.t.

$$\begin{aligned} 1 - y_{ii} - y_{jj} + y_{ij} &= 0 \qquad \forall i,j \in E(G) \\ \mathbf{Y} &\in \text{PSD(n)} \end{aligned}$$

To see the motivation behind this is constraint, remember that y_{ij} are supposed to represent exactly, in the integral setting, the product $x_i x_j$. In matrix form, this would mean that:

$$\mathbf{Y} = \mathbf{x}^T\mathbf{x}$$

where $\mathbf{x}$ is the column vector formed by stacking the scalars $x_i,\ i \in V(G)$. This equality implies that Y is PSD.

4 Lift and project in the general setting

The example shown in the previous section illustrates the steps involved in the method:

1. Starting with an integer program, we build a quadratic program out of it.
2. We replace products with novel linear variables, thus **lifting the dimensionality**
3. We add constraints on these linear variables (e.g. symmetry in the case of the linear programming variant, or PSD-ness in the case of the semidefinite programming variant)
4. We solve the resulting system and we **project** back to the original lower dimensional space

Let us now show how these steps work in the general setting.

4.1 Homogenizing the constraints

We start with an integer program:

$$\min_{\mathbf{x}} \mathbf{c}^T \cdot \mathbf{x}$$

s.t.

$$\mathbf{A}\mathbf{x} \geq \mathbf{b}$$
$$\mathbf{x} \in \{0,1\}^n$$

In the subsequent endeavor, it will be convenient to homogenize the equations. We do this by introducing a supplementary variable x_0 which will always have the value 1 in a feasible solution. The domain becomes:

$$\mathbf{A}\mathbf{x} \geq \mathbf{b} \cdot x_0$$
$$\mathbf{x} \in \{0,1\}^n,\ x_0 = 1$$

Which can be rewritten in matrix form as:

$$\begin{bmatrix} \mathbf{A} & -\mathbf{b} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ x_0 \end{bmatrix} \geq 0$$
$$\mathbf{x} \in \{0,1\}^n,\ x_0 = 1$$

and further, using some notation for succinctness:

$$\begin{aligned} \mathbf{A}'\mathbf{x}' \geq 0 \\ \mathbf{x}' \in \{0,1\}^{n+1},\ x_0 = 1 \end{aligned}$$

Whenever we use $\mathbf{x}'$, we mean the column vector $\mathbf{x}$ concatenated with the value x_0.

4.2 Quadratic programming

We now rewrite the integer program in quadratic form. We start with the constraints $x_i \in \{0, 1\}$, which become:

$$x_i(x_0 - x_i) = 0,\ \forall i \in \{1, \ldots, n\} \tag{1}$$

Notice that we have used x_0 instead of the constant 1, to remain inside a homogeneous setting.

Let us now consider an arbitrary inequality from our domain, corresponding to the j^{th} line in the matrix $\mathbf{A}'$, namely:

$$\mathbf{a}_j^T\mathbf{x}' \geq 0,\ \forall j \in \{1, \ldots, n\}$$

We can safely multiply this constraint with x_i and $x_0 - x_i$ for any $i \in \{1, \ldots, n\}$, because both of these quantities are nonnegative. Hence we obtain:

$$\begin{aligned} x_i\mathbf{a}_j^T\mathbf{x}' &\geq 0,\ \forall i, j \in \{1, \ldots, n\} \\ (x_0 - x_i)\mathbf{a}_j^T\mathbf{x}' &\geq 0,\ \forall i, j \in \{1, \ldots, n\} \end{aligned}$$

Hence from n inequalities, we have obtained $2n$ quadratic inequalities.

4.3 The lift step

We can write the inequalities in our linear program as a summation over products of variables:

$$\sum_{k=0}^{n} a_{jk}x_ix_k \geq 0,\ \forall i, j \in \{1, \ldots, n\} \tag{2}$$

$$\sum_{k=0}^{n} a_{jk}(x_0x_k - x_ix_k) \geq 0,\ \forall i, j \in \{1, \ldots, n\} \tag{3}$$

This form allows us to relax this quadratic program and obtain a linear program, by introducing $2n + 1$ novel variables:

$$y_{ij} = x_ix_j,\ \forall i, j \in \{0, \ldots, n\}$$

Constraint (1) becomes:

$$y_{i0} = y_{ii},\ \forall i \in \{1, \ldots, n\} \tag{4}$$

while constraints (2) and (3) become:

$$\sum_{k=0}^{n} a_{jk} y_{ik} \geq 0,\ \forall i, j \in \{1, \ldots, n\}$$
$$\sum_{k=0}^{n} a_{jk}(y_{0k} - y_{ik}) \geq 0,\ \forall i, j \in \{1, \ldots, n\}$$

4.4 Additional constraints

Because of the meaning we have assigned to the variable y_{ij}, it makes sense to add the following constraints to our LP:

$$\begin{aligned} y_{ij} &= y_{ji} \\ x_i &= y_{0i} \end{aligned}$$

The first one implies the symmetry of the product, while the second encodes the constraint $x_0 = 1$. Hence, the final lifted version of our LP is:

$$\min_{\mathbf{x}} \mathbf{c}^T \cdot \mathbf{x}$$

s.t.

$$\begin{aligned} x_i = y_{0i} &= y_{ii}, && \forall i \in \{1, \ldots, n\} && (5) \\ y_{ij} &= y_{ji}, && \forall i, j \in \{0, \ldots, n\} && (6) \\ \sum_{k=0}^{n} a_{jk} y_{ik} &\geq 0, && \forall i, j \in \{1, \ldots, n\} && (7) \\ \sum_{k=0}^{n} a_{jk}(y_{0k} - y_{ik}) &\geq 0, && \forall i, j \in \{1, \ldots, n\} && (8) \\ x_0 &= 1, && && (9) \end{aligned}$$

4.5 The project step

The final step in the method is to project the constraint back into the space involving only the x_i variables. This is achieved by adding together linear combinations of inequalities (7) and (8) with positive coefficients, canceling out some of the terms and then replacing the remaining terms using (5). This will lead to a set of inequalities involving only the variables x_i.

To give a geometric intuition behind this step, one can visualize the lifted LP as being a polytope in a $2n + 1$ dimensional space. By building linear combinations of inequalities and cancelling out the y_{ij}, we are actually projecting the polytope onto the $n + 1$ dimensions corresponding to the $x_i = y_{0i}$ variables (see figure Figure 1).

We shall not develop further this step formally, but shall illustrate it with an example of vertex cover for the complete triangle and show that the odd cycle constraint is derived at the end of the project step. The quadratic program we obtain is shown below:

$$\min_{\mathbf{x}} x_1 + x_2 + x_3$$

s.t.

$$\begin{aligned} x_1 + x_2 &\geq x_0 \\ x_1 + x_3 &\geq x_0 \\ x_2 + x_3 &\geq x_0 \\ x_1(x_0 - x_1) &= 0 \\ x_2(x_0 - x_2) &= 0 \\ x_3(x_0 - x_2) &= 0 \\ x_0 &= 1 \end{aligned}$$

We can multiply inequalities in many ways (more precisely in $n^2 = 9$ ways). We shall write only a few of them which are needed to make our point, for succinctness:

$$\begin{aligned} (x_0 - x_1)(x_1 + x_2) &\geq (x_0 - x_1)x_0 \\ (x_0 - x_2)(x_2 + x_3) &\geq (x_0 - x_2)x_0 \\ (x_0 - x_3)(x_1 + x_3) &\geq (x_0 - x_3)x_0 \\ x_1(x_2 + x_3) &\geq x_1 x_0 \\ x_2(x_1 + x_3) &\geq x_2 x_0 \end{aligned}$$

After performing the multiplications, the lift step and simplifying equivalent terms, one obtains:

$$\begin{aligned} y_{01} + y_{02} - y_{12} &\geq y_{00} \\ y_{02} + y_{03} - y_{23} &\geq y_{00} \\ y_{01} + y_{03} - y_{13} &\geq y_{00} \\ y_{12} + y_{13} - y_{01} &\geq 0 \\ y_{12} + y_{13} - y_{02} &\geq 0 \end{aligned}$$

By muliplying the first inequality by 2 and adding together, one obtains:

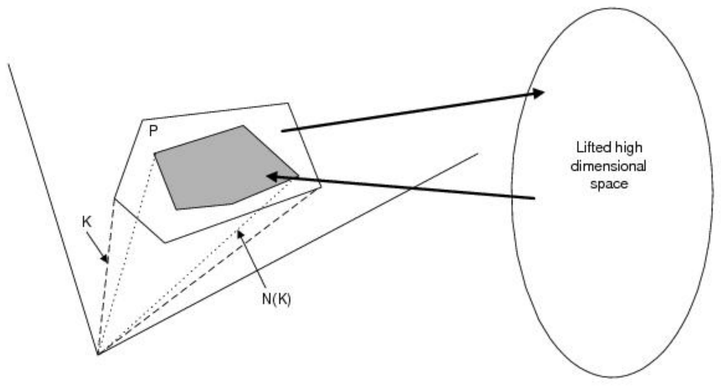

Figure 1: An intuitive diagram for the lift and project method. P is the original polytope, K is the cone obtained after homogenizing the variables (which includes P), $N(K)$ is the projected cone (entirely included in K).

$$2y_{01} + 2y_{02} + 2y_{03} \geq 4y_{00}$$

Which is obviously equivalent to the odd cycle constraint:

$$x_1 + x_2 + x_3 \geq 2$$

Hence we have shown that, among other inequalities, the odd-cycle constraint is derived in the project step. This result can be generalized to any graph, not just the triangle, and implies that one iteration of lift an project generates a relaxation that is at least as strong as that obtained by using the odd-cycle constraints alone.

5 A formal definition of the projected domain

Having presented the method, it might be interesting to provide a formal definition of the new projected domain in terms of the original one. Let us denote by $P \in \mathbb{R}^n$ the original polytope (corresponding to $\mathbf{Ax} \geq 0$) and by $K \in \mathbb{R}^{n+1}$ the cone obtained by homogenizing the constraints (corresponding to $\mathbf{A}'\mathbf{x}' \geq 0$). The result of lift-and-project will be another cone $N(K) \in \mathbb{R}^{n+1}$, living in the same space and K (see figure Figure 1).

The following definition characterizes the points in $N(K)$:

Definition 5.1. A point $\mathbf{y}$ is in the cone $N(K) \in \mathbb{R}^{n+1}$ if and only if there exists a matrix $\mathbf{Y} \in \mathbf{R}^{(n+1)\times(n+1)}$ such that:

1. $\mathbf{Y}$ is symmetric

2. $\mathbf{Y}\mathbf{e}_0 = \text{diag}(\mathbf{Y}) = \mathbf{y}$
3. $\mathbf{Y}\mathbf{e}_i \in K$, $\mathbf{Y}(\mathbf{e}_0 - \mathbf{e}_i) \in K, \forall i \in \{1, \ldots, n\}$

This definition actually captures the process in which we have derived constraints troughout the method. For each projected point $\mathbf{y}$ in the resulting cone, the matrix $\mathbf{Y}$ is representing the lifted variables corresponding to that point. We shall mention the intuition behind each of the properties in the definition:

1. We have imposed symmetry explicitly in the lifted problem, so obviously $\mathbf{Y}$ must obey this constraint too.
2. This requirement is saying that the zeroth column of $\mathbf{Y}$ is equal to its diagonal both vectors contain projected variables. This requirement has also been explicitly added to the lifted LP by imposing $x_i = y_{i0} = y_{ii}$.
3. This last requirement captures the way we have derived our constraints. Let us consider an arbitrary solution that $\mathbf{y} \in N(K)$. We can write a constraint:

   $$y_i \mathbf{a}_j^T \mathbf{y} \geq 0$$

   in matrix form as:

   $$\mathbf{a}_j^T \mathbf{y}\mathbf{y}^T \mathbf{e}_i \geq 0$$

   But if we note that $\mathbf{Y}$ is representing $\mathbf{y}\mathbf{y}^T$, we have that:

   $$\mathbf{a}_j^T \mathbf{Y}\mathbf{e}_i \geq 0$$

   But this inequality is equivalent to stating that $\mathbf{Y}\mathbf{e}_i \in K$. A similar intuition lies behind the constraint $\mathbf{Y}(\mathbf{e}_0 - \mathbf{e}_i) \in K$

We can also extends this definition to the case of semidefinite programming, by imposing an additional constraint:

Definition 5.2. A point $\mathbf{y}$ is in the cone $N_+(K) \in \mathbb{R}^{n+1}$ if and only if there exists a matrix $\mathbf{Y} \in \mathbf{R}^{(n+1)\times(n+1)}$ such that it obeys all the constraints imposed by definition Definition 5.1 plus one additional constraint, namely:

1. $\mathbf{Y}$ is PSD

5.1 Showing that this is a relaxation of the original problem

In order for the domain $N(K)$ to be a relaxation of the domain K, we have to show that $N(K)$ does not clip away any integral solution.

Claim 5.3. *The domain $N(K) \subset K$ contains all the integral solutions contained in K.*

Proof. Let $\mathbf{y} \in \{0,1\}^{n+1}$ be an integral solution from K. Let us consider the matrix $\mathbf{Y} = \mathbf{y}\mathbf{y}^T$. It is easy to see that this matrix obeys all the constraints in the definition of $N(K)$. Indeed, the matrix $\mathbf{Y}$ is symmetric by definition. The diagonal elements are $y_{ii} = y_i^2 = y_i$ (because $y_i \in \{0,1\}$), and the elements in the first column are $y_{0i} = y_0 y_i = y_i$ (because $y_0 = 1$), so the second property holds too. As for the last property, let us consider one arbitrary column of Y:

$$\mathbf{Y}\mathbf{e}_i = \begin{bmatrix} y_i y_0 \\ y_i y_1 \\ \cdots \\ y_i y_n \end{bmatrix} = y_i \cdot \mathbf{y} \in K$$

This follows because $\mathbf{y} \in K$ and K is a cone. Similarly, it follows that $\mathbf{Y}(\mathbf{e}_0 - \mathbf{e}_i)$ is in the cone.

Hence we have built a matrix $\mathbf{Y}$ that obeys the constraints in the definition, hence it follows that $\mathbf{y} \in N(K)$ q.e.d.

□

We can also show the same result for the tighter domain $N_+(K)$, i.e.:

Claim 5.4. *The domain $N_+(K) \subset N_K \subset K$ contains all the integral solutions contained in K.*

Proof. The only constraint that remains to be checked is the PSD-ness constraint. This follows trivially, since the matrix $\mathbf{Y}$ which we have constructed in the previous proof is PSD by definition.

□

5.2 Some complexity results

We succinctly mention some complexity results for this method. The lift-and-project operator $N(K)$ (or $N_+(K)$ if one works in the semidefinite programming setting) can be applied recursively any number of times:

$$N^r(K) = N(N^{r-1}(K))$$

We provide two interesting complexity properties that arise from this procedure (for formal proofs, please refer to [LS91]):

Claim 5.5. *If K has a poly-time separation oracle, the one can optimize a linear function over $N^r(K)$ in time $n^{O(r)}$.*

Claim 5.6. *If the original problem has n variables, then $N^n(K) = K_0$ where K_0 denotes the integral hull of the integer problem.*

Notice that the latter results does not help us solve the problem efficiently, since the time required to obtain the integral hull would be $n^{O(n)}$.

Hence we conclude that the interesting problem to look at now in the field is to determine for which problems a few rounds of method application give some non-trivial improvement over the original relaxation.

Another issue still under investigation is the effect of lift-and-project on the integrality gap for different classes of problems. One negative result for Vertex cover can be found in [GMPT06].

References

[GMPT06] Konstantinos Georgiou, Avner Magen, Toniann Pitassi, and Iannis Tourlakis. Integrality gaps of 2-o(1) for vertex cover sdps in the lovasz-schrijver hierarchy. *Manuscript*, 2006.

[LS91] Laszlo Lovasz and Alexander Scrijver. Cones of matrices and set-functions, and 0-1 optimization. *SIAM Journal on Optimization*, 1:166–190, 1991.

[Tou06] Iannis Tourlakis. *New lower bounds for Approximation Algorithms in the Lovasz-Schrijver Hierarchy*. PhD thesis, Princeton University, 2006.