IE 511: Integer Programming, Spring 2021 **16 Mar, 2021**

Lecture 15: NP-completeness, Optimization and Separation

Lecturer: Karthik Chandrasekaran *Scribe: Karthik*

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications.*

We were interested in understanding "efficiency of algorithms". For the sake of this discussion, it is helpful to focus on decision problems—we already saw the relevant decision version of optimization problems. We saw that algorithms whose run time is polynomial in the size of the input are deemed to be efficient algorithms. Next, we saw the family of problems NP and we began discussing the relative difficulty of problems with respect to each other.

Recap

Definition 1. NP is the family of decision problems whose YES instances have a polynomial-sized certificate.

Definition 2. A *polynomial-time* reduction from a problem π to a problem π' is a polynomial-time computable function f such that S is a YES instance of problem π iff $f(S)$ is a YES instance of problem π'.

Note that if we have a polynomial-time reduction from problem π to problem π', then a polynomial time algorithm for problem π' would immediately allow us to also solve problem π.

Example of a polynomial-time reduction:

π:	Min Cardinality Vertex Cover
Given:	A graph $G = (V, E), k \in \mathbb{Z}_+$
Goal:	Does there exist a vertex cover in G of size $\leq k$?
Recall:	$U \subseteq V$ is a *vertex cover* in G if every edge in G has at least one vertex in U.
π':	Max Cardinality Stable set
Given:	A graph $G = (V, E), r \in \mathbb{Z}_+$
Goal:	Does there exist a stable set in G of size $\geq r$?
Recall:	A set $S \subseteq V$ is a *stable set* in G if no edge of G has both its end vertices in S.

Proposition 3. *There exists a polynomial-time reduction from π to π' and vice-versa.*

Proof. $\underline{\pi \to \pi'}$:

Given an instance of vertex cover problem (G, k), consider the mapping $f(G, k) = (G, |V(G)| - k)$. Then

$$\begin{aligned} & G \text{ has a vertex cover } U \text{ of size at most } k \\ \iff & G - U \text{ has no edges} \\ \iff & V(G) - U \text{ is a stable set in } G \\ \iff & G \text{ has a stable set of size at least } |V(G)| - k. \end{aligned}$$

The mapping f is polynomial-time computable.

$\underline{\pi' \to \pi}$:

Given an instance of stable set (G, r), consider the mapping $f(G, r) = (G, |V(G)| - r)$. Then,

$$\begin{aligned} & G \text{ has a stable set } S \text{ of size at least } r. \\ \iff & \text{All edges of } G \text{ are between vertices in } V(G) - S \text{ and } V(G). \\ \iff & V(G) - S \text{ is a vertex cover in } G \text{ of size at most } |V(G)| - r. \\ \iff & G \text{ has a vertex cover } U \text{ of size at most } |V(G)| - r. \end{aligned}$$

The mapping f is polynomial-time computable. □

Proposition 3 tells us that for the purpose of polynomial-time algorithms, the problems min cardinality vertex cover and max cardinality stable set are equivalent: solving one in polynomial-time will allow us to solve the other also in polynomial-time.

The family of most difficult problems (difficult in terms of computation time) are known as NP-complete problems. We have the following formal definition:

Definition 4. A problem π in NP is *NP-complete* if every problem $\pi' \in$ NP has a polynomial-time reduction to π (see Figure 15.1).

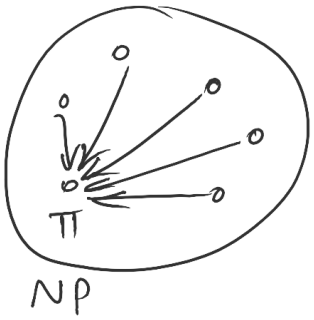

Figure 15.1: All problems in NP reduce to the problem π. Then, π is said to be NP-complete.

Significance of NP-complete problems. If there exists a polynomial-time algorithm for an NP-complete problem, then we have a polynomial-time algorithm for all problems in NP. So it is sufficient to design polynomial-time algorithms for NP-complete problems.

How do we show that a problem π' is NP-complete? Suppose problem π is NP-complete. If (1) $\pi' \in NP$ and (2) there exist a polynomial-time reduction from π to π', then π' is also NP-complete (see Figure 15.2).

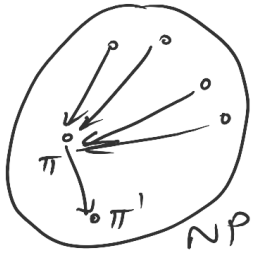

Figure 15.2: All problems in NP reduce to the problem π and the problem π reduces to π', where $\pi, \pi' \in NP$. Then π' is also NP-complete.

We have the following notion related to NP-completeness for optimization problems:

Definition 5. An optimization problem whose decision version is NP-complete is called *NP-hard*.

Question. We defined the notion of NP-complete problems, but is there an example? Does there exist even one NP-complete problem?

[Cook] Yes, Satisfiability problem is NP-complete.

15.1 Satisfiability (SAT)

We now discuss the Satisfiability problem.

Given: n Boolean variables $z_1, \ldots, z_n$,
m clauses $C_i = L_{i_1} \vee L_{i_2} \vee \cdots \vee L_{i_j}$ where $L_{i_k} \in \{z_{i_k}, \overline{z}_{i_k}\}$.
Goal: Does there exist an assignment of $\{0, 1\}$ to variables $z_1, \ldots, z_n$ such that all clauses evaluate to 1?

Here, $\vee$ corresponds to Boolean OR.

Example: Consider a SAT instance with variables: z_1, z_2, z_3 and clauses:

$$C_1 = z_1 \vee z_2 \vee \bar{z}_3$$
$$C_2 = \bar{z}_1 \vee z_2$$
$$C_3 = \bar{z}_1 \vee \bar{z}_2 \vee z_3.$$

Then, $z_1 = 1, z_2 = 1, z_3 = 1$ is a satisfying assignment.

Note: SAT is in NP.

The following result is fundamental to the theory of efficient algorithms/computational complexity.

Theorem 6 (Cook)**.** *SAT is NP-complete.*

Theorem 6 says that every problem in NP has a polynomial-time reduction to SAT. Hence, SAT is a difficult problem in NP and if we have a polynomial-time algorithm to solve SAT, then we will solve all problems in NP in polynomial-time.

Next, where does integer programming problem sit relative to NP and NP-complete problems? Let us focus on binary integer programs.

Recap: Decision-BIP

Given: $A \in \mathbb{Z}^{m\times n}, b \in \mathbb{Z}^m$
Goal: Does there exist $x \in \{0,1\}^n$ such that $Ax \leq b$?

The following result shows that Decision-BIP is a difficult problem among the problems in NP.

Theorem 7. *Decision-BIP is NP-complete.*

Proof. (i) Decision-BIP is in NP:

Suppose an instance (A, b) has a feasible $\bar{x}$. Then $\bar{x}$ is a polynomial-sized certificate because $\text{size}(\bar{x}) = n$.

(ii) Decision-BIP is NP-complete:

We need to show that there exists an NP-complete problem π such that π is polynomial time reducible to Decision-BIP. We will show that SAT reduces to Decision-BIP.

Let (vars: $z_1, \ldots, z_n$, clauses: $C_1, \ldots, C_m$) be an instance of SAT. Consider the mapping f(vars: $z_1, \ldots, z_n$, clauses: $C_1, \ldots, C_m$) that gives the following BIP instance:

$$\sum_{i\in[n]:z_i\in C_j} x_i + \sum_{i\in[n]:\bar{z}_i\in C_j} (1 - x_i) \geq 1 \quad \forall j \in [m]$$

$$x_i \in \{0,1\} \quad \forall i \in [n].$$

Let A be the constraint matrix, b be the RHS. Then, $A \in \{0, 1, -1\}^{m\times n}$, $b \in \{1, -1, 0, \ldots, -(n-1)\}^m$. Therefore, size$(A, b)$ = poly(size(SAT instance)) and z is a satisfying assignment for the SAT instance iff $x = z$ satisfies the constraints of the resulting Decision-BIP instance. Therefore, the SAT instance $(z_1, \ldots, z_n, C_1, \ldots, C_m)$ is a YES instance iff the Decision-BIP instance $f(z_1, \ldots, z_n, C_1, \ldots, C_m)$ is a YES instance. We thus have a polynomial-time reduction f from SAT to Decision-BIP.

□

Corollary 7.1. *BIP is NP-hard. MIP and IP are NP-hard.*

Note: BIP can be reduced to IP and MIP.

To show NP-completeness of decision variants of IP and MIP, we need to also show that they are in NP. Showing that decision variants of MIP and IP are in NP is non-trivial, but nevertheless true. So, *Integer Programming Problem*, the main focus of this course, is a difficult problem among the problems in NP. I.e., if we have a polynomial time algorithm for IP, then we will have polynomial time algorithms for all problems in NP.

15.2 Optimization and Separation

We were alluding to connections between efficient optimization and *efficient separation*. Now that we understand what it means to be efficient, let us make this connection between optimization and separation precise.

Recall that we saw linear programs which had exponential number of constraints, e.g, consider max weight forest problem, which is a special case of the matroid optimization problem:

Recap: LP-relaxation for the max weight forest problem

$$\begin{aligned} \max & \sum_{e \in E} w_e x_e \\ \text{s.t.} & \sum_{e \in F} x_e \leq r(F) \quad F \subseteq E \\ & x_e \geq 0 \quad \forall e \in E \end{aligned} \tag{15.1}$$

where $r : 2^E \to \mathbb{Z}_{\geq 0}$ is the rank function of the graphic matroid on the graph $G = (V, E)$.

We saw that every extreme point optimal solution to this LP will be integral and correspond to the indicator vector of a max weight forest. But how do we solve this LP? How can we solve LPs in which the number constraints is very large (very large relative to the number of variables)? In the above example, the number of constraints is exponential in the number of variables, i.e., exponential in the size of input—recall that the input consists of a graph with weights on edges.

Geometric Intuition: We may not need to write down all constraints of the LP to be able to solve it efficiently along a particular direction. See Figure 15.3.

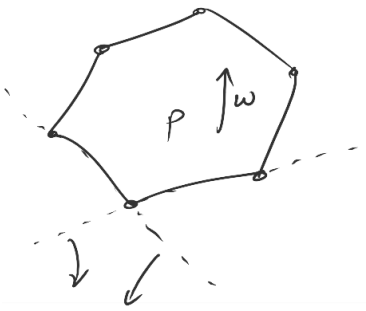

Figure 15.3: The constraints marked with arrows are irrelevant along the objective direction of interest, namely w.

There is a clean result that translates this intuition towards a polynomial time algorithm for solving LPs with large number of constraints. In order to describe this intuition, we consider optimizing over an implicitly given polyhedron.

Implicitly represented polyhedron. A polyhedron associated with a linear/integer programming problem could be represented in an implicit fashion or an explicit fashion: for example, the polyhedron described by the constraints of (15.1) can be represented (i) either by explicitly writing down all constraints for the graph G of interest (ii) or by simply specifying the graph G. The graph G is an implicit representation. Note that specifying the graph G allows you to infer all constraints and this is why, this representation is known as implicit representation. The advantage of the implicit representation is that it is small-sized. In contrast, the explicit representation blows up in size (becomes exponential in the size of the graph).

The polyhedron associated with several discrete optimization problems have this property: they have an implicit representation that is small-sized, but the explicit representation could become exponential in the size of the input. It is convenient to consider the implicit representation when dealing with such polyhedra.

We consider optimizing over implicitly defined polyhedra.

Optimization Problem (over an implicitly given polyhedron):

Given: A bounded rational polyhedron $P \subseteq \mathbb{R}^n$ (given implicitly) and a rational vector $w \in \mathbb{R}^n$
Goal: Verify if P is empty and if so, find $x^* \in P$ maximizing $w^T x$.

The optimization problems turns out to be closely related to the *separation problem*.

Separation Problem (over an implicitly given polyhedra):

Given: A bounded rational polyhedron $P \subseteq \mathbb{R}^n$ (given implicitly) and a rational vector $\bar{x} \in \mathbb{R}^n$
Goal: Verify if $\bar{x} \in P$ and if not, then find a rational vector $c \in \mathbb{R}^n$ and a rational value $\delta \in \mathbb{R}$ such that

$$c^T x < \delta \quad \forall x \in P \quad \text{and} \quad c^T \bar{x} > \delta.$$

Optimization is equivalent to Separation. A landmark result in optimization shows that the optimization problem and the separation problem are equivalent.

Theorem 8 (Grotschel-Lovasz-Schrijver)**.** *[informal version] The separation problem is polynomial-time solvable iff the optimization problem is polynomial-time solvable.*

The proof of GLS' result is via the *Ellipsoid algorithm*. We will not cover the ellipsoid algorithm in this course.

Remark. GLS' result is a proof of concept. It gives an efficient algorithm for the optimization problem if the separation problem is solvable efficiently but the resulting algorithm for solving the optimization problem is not the fastest algorithm to solve it. GLS' result should be seen as an encouraging sign in the search for efficient algorithms for the optimization problem. Upon encountering an algorithm for the optimization problem thorough an algorithm for the separation problem, one should look for direct algorithms to solve the optimization problem, i.e., study the problem further to find fast oalgorithms.

15.3 Linear Equations Integer Feasibility Problem (LEIF)

We now consider another special case of integer programs—namely, Linear Equations Integer Feasibility Problem (LEIF). Given a constraint matrix $A \in \mathbb{Z}^{m \times n}$, and a RHS vector $b \in \mathbb{Z}^m$, we have the following problems:

Linear equation:	Does there exist $x \in \mathbb{R}^n : Ax = b$?	(Gaussian Elimination)
Linear program:	Does there exist $x \in \mathbb{R}^n : Ax \leq b$?	(Ellipsoid Algorithm)
Integer program:	Does there exist $x \in \mathbb{Z}^n : Ax \leq b$?	(NP-complete)
LEIF:	Does there exist $x \in \mathbb{Z}^n : Ax = b$?	(?)

Note that Linear equation admits an efficient algorithm, namely Gaussian Elimination; Linear program also admits an efficient algorithm, namely the Ellipsoid algorithm; in contrast, Integer program is NP-complete. How about LEIF? Is it also NP-complete or does it admit an efficient algorithm?

Let us see an example to understand how we would go about solving LEIF.

Example:

$$\begin{bmatrix} 2 & 1 & 6 & 4 \\ 7 & 2 & 5 & 5 \\ 8 & 3 & 33 & 10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} \tag{15.2}$$

Does there exist $x_1, x_2, x_3, x_4 \in \mathbb{Z}$ satisfying system (15.2)?

We can conclude that it has no integral solution as follows:

$\exists x \in \mathbb{Z}^4$ satisfying system (15.2)

$$\iff \exists x \in \mathbb{Z}^4 : \begin{bmatrix} 1 & 6 & 2 & 4 \\ 2 & 5 & 7 & 5 \\ 3 & 33 & 8 & 10 \end{bmatrix} x = \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}$$

$$\iff \exists y \in \mathbb{Z}^4 : \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & -7 & 3 & -3 \\ 3 & 15 & 2 & -2 \end{bmatrix} y = \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} \qquad \begin{cases} c_2 \leftarrow c_2 - 6c_1 \\ c_3 \leftarrow c_3 - 2c_1 \\ c_4 \leftarrow c_4 - 4c_1 \end{cases}$$

$$\iff \exists x \in \mathbb{Z}^4 : \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & -1 & 3 & 0 \\ 3 & 19 & 2 & 0 \end{bmatrix} x = \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} \qquad \begin{cases} c_2 \leftarrow c_2 + 2c_3 \\ c_4 \leftarrow c_4 + c_3 \end{cases}$$

$$\iff \exists x \in \mathbb{Z}^4 : \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & -19 & 59 & 0 \end{bmatrix} x = \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} \qquad \begin{cases} c_2 \leftarrow -c_2 \\ c_3 \leftarrow c_3 + 3c_2 \end{cases}$$

$\iff \exists x \in \mathbb{Z}^4 :$

$x_1 = 1$

$2x_1 + x_2 = 3, \quad$ i.e., $x_2 = 1$

$3x_1 - 19x_2 + 59x_3 = 1, \quad$ i.e., $59x_3 = 17 \implies$ No integral solution

Therefore, there is no integral solution for system (15.2).

The first $\iff$ is obvious since it is equivalent to relabeling the variables. Here is a brief argument for why the second $\iff$ holds: Suppose that there exists $x \in \mathbb{Z}^4$ satisfying the first system. Then, set $y_1 = x_1 + 6x_2 + 2x_3 + 4x_4, y_2 = x_2, y_3 = x_3, y_4 = x_4$ to obtain an integral solution for the second system. Similarly, if there exists $y \in \mathbb{Z}^4$ satisfying the second system, then setting $x_1 = y_1 - 6y_2 - 2y_3 - 4y_4, x_2 = y_2, x_3 = y_3, x_4 = y_4$ gives an integral integral solution for the first system. The arguments for the remaining $\iff$ are similar.

Note that each individual step above holds because we are adding an integral multiple of some column to another column or we are negating the sign of a column or we are exchanging columns. In particular, we are never dividing any column by an integer in our column operations.

In the next lecture, we will formalize the ideas underlying our solution for the example to see that LEIF admits an efficient algorithm.