18.433 Combinatorial Optimization

NP-completeness

November 18 *Lecturer: Santosh Vempala*

Up to now, we have found many efficient algorithms for problems in Matchings, Flows, Linear Programs, and Convex Programming. All of these are polynomial-time algorithms.

But there are also problems for which we have found no polynomial-time algorithms. The theory of NP-completeness unifies these failures. Roughly speaking, an NP-complete problem is one that is as hard as any problem in a large class of problems. For example, the Traveling Salesman Problem (TSP), Integer Programming (IP), the Longest Cycle, and Satisfiability (SAT) are all hard problems. NP-completeness tells us that they are all, in a precise sense, equally hard. Let's look at each problem in a little more detail.

1. The Traveling Salesman Problem

   Let's say that there exist a salesman that has to visit n cities and there exists a distance $w_{i,j}$ between cities i and j. He wants to make sure to minimize his traveling time by visiting every city exactly once. In other words, there is a complete graph $G = (V, E)$ with lengths $w_{i,j}$ between nodes i and j. The question we must ask is: What is the shortest cycle that visits every node exactly once?

2. Integer Linear Programming

   Suppose that you have a linear program such as the following:

   $$\min c^T x$$
   $$Ax \leq b \text{ for } x_i \geq 0$$

   This is your typical linear program. Now, if you decide to add an integrality constraint on x_i such that it is forced to be a positive integer, then you have an Integer Linear Program (ILP).

3. Boolean Satisfiability

   The satisfiability problem (SAT) uses boolean expressions such as the following

   $$f = (x_1 \vee x_2 \vee x_4) \wedge (x_3 \vee \bar{x}_4)....$$

with $x_i = \{\text{True,False}\}$ and using well known boolean identities. Does F have a satisfying assignment? Can we find values of x_i such that every clause in F is equal 1?

4. Longest Cycles

   Given a graph $G = (V, E)$, find the longest cycle.

5. Cliques

   A clique is a complete subgraph. Given a graph $G = (V, E)$, find a clique of maximum cardinality (vertices).

As different as these examples might seem, they have two main properties in common:

A) None of them is known to have a polytime algorithm.

B) If any one of them has a polytime algorithm, then they all do.

1 Optimization vs Decision

While property A seems trivial to us all by the inspection of each problem, property B is not as easy to see. To understand this property, we first formulate the *decision* versions of these optimization problems.

Find the optimum among a set of feasible solutions F with cost function c

vs

Is there a feasible solution of $cost \leq L$?

If the Optimal (OPT) is solved, then the Decision (DEC) is also solved. Namely, DEC reduces to OPT. Now, is OPT reducible to DEC? Well, using the TSP as an example, we ask: Is there a $tour \leq L$? Then, we proceed to do a binary search in order to find the length of the shortest tour, say S. But, we still don't know what the tour is. One way to figure this out is to use the following algorithm:

Take out an edge e.

Ask if the same graph still has a $tour \leq S$.

If it does, then we don't need that edge and can delete it.

If it doesn't, then we keep that edge because it will be part of our tour.

Repeat this algorithm for all the edges.

In the ILP example, we can ask: Is there an x of cost $\leq L$? Well, one way to do this would be to set $x_i = 0$ and if optimum stays the same, then we can fix that particular x_i to 0.

For the maximum cliques problem, the OPT problem would be: Find the largest clique, while the DEC problem would be: Is there a clique of size $\leq k$? To find the optimal size k^*, again we do a binary search. We then consider the graph with v_i and all its neighbors. If the optimum in this graph remains the same, then save that vertex, then we can delete all other vertices. Else, delete v_i because it is not in the max clique.

2 P and NP

2.1 Definitions

P : Class of decision problems that can be solved in polytime.

NP : Decision problems that have a short proof (certificate) for YES answers. The proof has length bounded by a polynomial in the size of the input, and its correctness can be verified in polytime.

Note that problems in P have short proofs for both YES and NO answers. This means that $P \subseteq NP$. Let's look at a problem in P:

Linear Programming: Is the minimum less than some c?

YES: Give a feasible solution $\leq c$

NO: Use the Dual of the problem to give a lower bound.

Now, let's look at the following examples of NP problems:

1. TSP, Is there a *tour* $\leq L$?

   YES: Give a tour

   NO: ?

2. SAT, Does there exist a satisfying assignment?

   YES: Give a satifying assignment

NO: ?

3. Min ILP, Is the minimum $\leq c$?

   YES: Give a feasible solution that is $\leq c$

   NO: ?

This leads to the question: Is $P = NP$?

2.2 Reductions

A reduction from a problem A to a problem B is a function $f : A \rightarrow B$ such that for all instances x

$$x \in A \Leftrightarrow f(x) \in B.$$

If the function f can be computed in polynomial time, then it is called a polynomial-time reduction. An implication of this is the following:

If there exists a polytime algorithm for B, then there exists one for A.

A problem B is NP-hard if every problem in NP has a polytime reduction to B. If, in addition, B is in NP, then it is NP-complete.

Thus if A is NP-complete, and it has a reduction to another problem B in NP, then B is also NP-complete.

2.3 Examples of Reduction

SAT is NP-complete (we will not prove this in class).

1. ILP is NP-complete Let's take the following SAT problem and see if it can be solved by an ILP.

   $$F = (x_1 \vee x_2 \vee ... \vee \bar{x}_i) \wedge (x_4 \vee \bar{x}_5) \wedge ... \wedge (x_a \vee x_b \vee ... \vee x_c)$$

   This SAT problem can also be written in the following way

   $$x_1 + x_2 + ... + \bar{x}_i \geq 1$$

   $$x_4 + \bar{x}_5 \geq 1$$

   $$x_a + x_b + ... + \bar{x}_c \geq 1$$

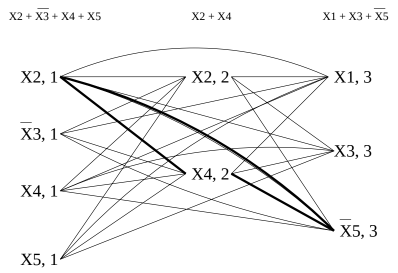

Figure 1: clique is NP-complete

$$\ldots$$

$$x_i = \begin{cases} 1, & \text{then true} \\ 0, & \text{then false} \end{cases}$$

Since SAT can be reduced to an ILP, ILP is NP-complete.

2. Clique is NP-complete

   SAT can be reduced to clique by the following construction. Suppose we have a formula F with m clauses.

   1) Vertices are going to be of the form $< x_a, i >$ where x_a is a literal that occurs in clause C_i

   2) Edges are going to be of the form $\{< x_a, i >, < x_b, j >\}$ for all $x_a \neq \bar{x}_b$ and $i \neq j$.

   By defining the vertices and edges this way, we ensure that all the connected vertices are compatible, since their truth values won't overlap. If we find a clique of size m in this graph, F is satisfiable. Refer to Fig. 1.