15-451 Algorithms, Spring 2017
Recitation #9 Worksheet

NP-Completeness Reductions (general). To show that a problem B is NP-Complete, we take a *known NP-Complete* problem A, and then we reduce A to B. I.e., we show that $A \leq_p B$. We do this by coming up with a polynomial-time procedure f for taking instances x of problem A and converting them to instances $f(x)$ of problem B such that $f(x)$ is a YES-instance of B *if and only if* x is a YES-instance of A. Make sure you understand:

- Why do we reduce this way, and not the other way around?
- Why is the *if and only if* condition important? Why wouldn't this work if f only satisfied the "if" or "only if"?

Binary LPs. Binary linear programming (BinLP) is like linear programming, with the additional constraint that all variables must take on values either 0 or 1. The decision version of binary linear programming asks whether or not there exists a point satisfying all the constraints. (For the decision version there is no objective function).

Show that BinLP is NP-complete.

- Show that BinLP is in NP.

  Solution: What is the witness? The solution.

- Reduce a NP-hard problem to BinLP. (Remember, you should use a Karp reduction, and the reduction should take polynomial time.)

  Solution: We can reduce 3SAT to BinLP. Given an instance I of 3SAT, let the variables in ϕ be $x_1, x_2, ..., x_n$. We produce an instance $f(I)$ of BinLP as follows: we have corresponding variables $z_1, z_2, ..., z_n$ in our BinLP. First, each variable is binary (either 0 or 1):

  $$z_i \in \{0, 1\} \qquad \forall i.$$

  Assigning $z_i = 1$ in the integer program represents setting $x_i = T$ in the formula, and assigning $z_i = 0$ represents setting $x_i = F$. Now for each clause like $(x_1 \vee \bar{x_2} \vee x_3)$, we have a constraint like:

  $$z_1 + (1 - z_2) + z_3 \geq 1.$$

  To satisfy this inequality we must either set $z_1 = 1$ or $z_2 = 0$ or $z_3 = 1$, which means we either set $x_1 = T$ or $x_2 = F$ or $x_3 = T$ in the corresponding truth assignment. More generally, for each clause in the 3SAT instance, we create the constraint that the sum of literals, using z_i to represent x_i and $(1 - z_i)$ to represent $\bar{x_i}$, is at least 1.

  If the given instance I was a YES-instance of 3SAT then $f(I)$ is a YES-instance for BinLP: just take a satisfying assignment A to the variables x_i and set each z_i to 0 or 1 accordingly. Since A satisfied at least one literal in each clause, this means the associated sum is ≥ 1.

In the other direction, any solution to the BinLP must set at least one of the associated literals to 1, since each is an integer 0 or 1.

Finally, the transformation is clearly poly time.

Integer LPs. Integer linear programming (ILP) is like linear programming, with the additional constraint that all variables must take on values in the integers $\mathbb{Z}$. The decision version of integer programming asks whether or not there exists a point satisfying all the constraints. (Again for the decision version there is no objective function). Note that the above reduction, with a small tweak, immediately shows that ILP is NP-hard. Do you see why?

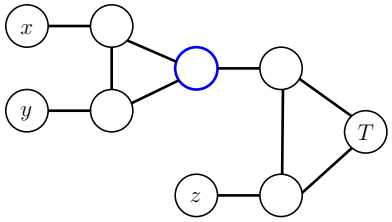
Solution: Just add the constraints $0 \leq z_i \leq 1$ for all i. BTW, membership in NP is a bit trickier here (how do you know that if the answer is YES, there is always a solution that can be described in a polynomial number of bits) but it follows from facts about matrices that we won't get into.

3-Coloring is NP-complete.

Some of you have seen a slightly different reduction from Circuit-SAT to 3-Coloring in 15-251. Here we'll reduce from 3SAT.

1. Step I: Why is 3-Coloring in NP?
2. Step II: We want to reduce 3SAT to 3-Coloring. Given a 3-CNF formula I, and we to produce a graph $G = f(I)$ such that G is 3-Colorable if and only if I is satisfiable.
   (a) Let's call the three colors R (red), T and F, and add three special nodes in a triangle called R, T, and F that we can assume without loss of generality are given the corresponding colors.
   (b) For each x_i, we have one node called x_i and one node called $\neg x_i$. Add a triangle between R, x_i, and $\neg x_i$ for each i. This forces the coloring to make a choice for each variable of whether it should be T or F.
   (c) Now, we need to add in a "gadget" for each clause. Say for $(x \vee y \vee z)$, we want to make it impossible to color all three of x, y, z with color F, but all other settings of $\{T, F\}$ are OK.
   Can you create such a gadget?

   Solution:



(Look at the triangle attached to x, y. If both $x = y = F$, then the tip of that triangle has to be F too, else it can be colored T. A similar argument now holds for the second triangle too.)

Once all these gadgets are added, a 3-Coloring exists of G if and only if there is a satisfying assignment to the original instance I. Finally, the reduction takes linear time: putting down this gadget for each clause in I.

k-Coloring is NP-complete. Can you show that 4-coloring is NP-complete? k-coloring for constant $k \geq 3$? What about 2-coloring?

Solution: Again k-coloring is in NP, just take the coloring and check it is valid, that each edge has distinct colors. For 4-coloring, reduce from 3-coloring. Take an instance I of 3-coloring, which is a graph. Take a new node and attach it to all the nodes of G, call this graph $H = f(I)$. This graph is 4-colorable if and only if G was 3 colorable. Hence,

$$I \text{ is a YES instance } \iff f(I) \text{ is a YES instance .}$$

Also, this reduction takes linear time: copying the graph G over, and adding a new vertex, connecting it to all other vertices.

You can use the same idea to show that k-coloring, for any constant k, is NP-complete.

2-coloring is in P: a graph is 2-colorable if and only if it is bipartite. This can be checked in linear time using DFS.

For the set cover analysis, see the lecture notes.