Notes for Lecture 17

1 Proving NP-completeness Results

In order to prove that an NP problem C is NP-complete we need to exhibit infinitely many reductions: we have to show that for every problem N in NP there is a reduction from N to C. In the next lecture, we will prove that a problem called CSAT (abbreviation of Circuit Satisfiability) is NP-complete.

Once we have found an NP-complete problem, however, proving that other problems are NP-complete becomes easier, since we now just need one more reduction.

Indeed, the following result clearly holds:

Lemma 1
If A reduces to B, and B reduces to C, then A reduces to C.

Proof: If A reduces to B, there is a polynomial time computable function f such that $A(x) = B(f(x))$; if B reduces to C it means that there is a polynomial time computable function g such that $B(y) = C(g(y))$. Then we can conclude that we have $A(x) = C(g(f(x))$, where $g(f())$ is computable in polynomial time. So A does indeed reduce to C. $\Box$

This immediately implies:

Lemma 2
Let C be an NP-complete problem and A be a problem in NP. If we can prove that C reduces to A, then it follows that A is **NP***-complete.*

Right now, literally thousands of problems are known to be **NP**-complete, and each one (except for a few "root" problems like CSAT) has been proved **NP**-complete by way of a single reduction from another problem previously proved to be **NP**-complete. By the definition, all **NP**-complete problems reduce to each other, so the body of work that lead to the proof of the currently known thousands of **NP**-complete problems, actually implies *millions* of pairwise reductions between such problems.

2 The Problem 3SAT

In the next lecture we will prove that 3-CNF Satisfiability, abbreviated 3SAT, is NP-complete. Now we define 3SAT and show how, once we know that 3SAT is NP-complete, we can prove the NP-completeness of several other problems.

An input of 3SAT problem is a Boolean formula over Boolean variables $x_1, \ldots, x_n$ in 3-Conjunctive Normal Form (3CNF). Conjunctive normal form means that the formula is a AND-of-OR, and the number 3 stands for the fact that each OR is over three variables or negated variables. For example the following is a 3CNF Boolean formula over Boolean variables x_1, x_2, x_3, x_4, x_5:

$$(x_2 \vee \bar{x}_4 \vee x_5) \wedge (x_1 \vee \bar{x}_3 \vee \bar{x}_4) \wedge (\bar{x}_2 \vee x_3 \vee x_5) \tag{1}$$

The symbol $\wedge$ stands for AND (looks a bit like an "a" as in "and") and the symbol $\vee$ stands for OR (looks like a "v", as in "vel" which is Latin for "or" – don't look for logic in the way logicians define notation). The bar over a variable stands for negation, so $\bar{x}_2$ should be read as NOT x_2.

Given a 3CNF Boolean formula, the goal of the 3SAT problem is to determine if there is an assignment of Boolean values to the Boolean variables that makes the formula evaluate to True (if this happens we say that the assignment *satisfies* the clause). For example, assigning every variable to True satisfies the formula in (1). A couple more useful pieces of terminology: the OR subformulas in a 3CNF formula (like $x_2 \vee \bar{x}_4 \vee x_5$) are called the *clauses* of the formula. A variable like x_2 and a negated variable like $\bar{x}_4$ are called *literals*. Thus a 3CNF formula is an AND of clauses, each clause is an OR of three literals, and each literal is either a variable or the negation of a variable.

We state the following result that we will prove later.

THEOREM 3
3SAT is NP-complete.

3 Some NP-complete Graph Problems

3.1 Independent Set

Given an undirected non-weighted graph $G = (V, E)$, an *independent set* is a subset $I \subseteq V$ of the vertices such that no two vertices of I are adjacent. (This is similar to the notion of a *matching*, except that it involves vertices and not edges.)

We will be interested in the following optimization problem: given a graph, find a largest independent set. We have seen that this problem is easily solvable in forests. In the general case, unfortunately, it is much harder.

The problem models the execution of conflicting tasks, it is related to the construction of error-correcting codes, and it is a special case of more interesting problems. We are going to prove that it is not solvable in polynomial time unless P = NP.

First of all, we need to formulate it as a decision problem:

- Given a graph G and an integer k, does there exist an independent set in G with at least k vertices?

It is easy to see that the problem is in NP. We have to see that it is NP-hard. We will reduce 3SAT to Maximum Independent Set.

Starting from a formula ϕ with n variables $x_1, \ldots, x_n$ and m clauses, we generate a graph G_ϕ with $3m$ vertices, and we show that the graph has an independent set with at least m vertices if and only if the formula is satisfiable. (In fact we show that the size of the largest independent set in G_ϕ is equal to the maximum number of clauses of ϕ that can be simultaneously satisfied. — This is more than what is required to prove the NP-completeness of the problem)

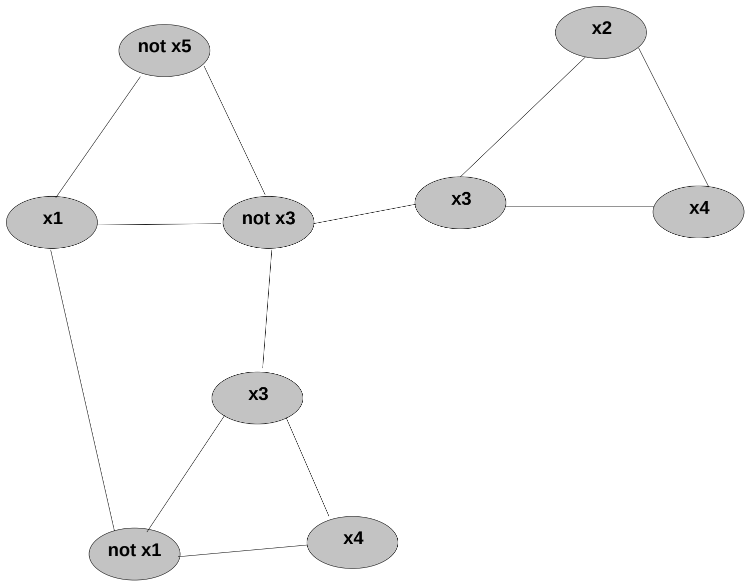

Figure 1: The reduction from 3SAT to Independent Set.

The graph G_ϕ has a triangle for every clause in ϕ. The vertices in the triangle correspond to the three literals of the clause.

Vertices in different triangles are joined by an edge iff they correspond to two literals that are one the complement of the other. In Figure 1 we see the graph resulting by applying the reduction to the following formula:

$$(x_1 \vee \bar{x}_5 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (x_3 \vee x_2 \vee x_4)$$

To prove the correctness of the reduction, we need to show that:

- If ϕ is satisfiable, then there is an independent set in G_ϕ with at least m vertices.
- If there is an independent set in G with at least m vertices, then ϕ is satisfiable.

From Satisfaction to Independence. Suppose we have an assignment of Boolean values to the variables $x_1, \ldots, x_n$ of ϕ such that all the clauses of ϕ are satisfied. This means that for every clause, at least on of its literals is satisfied by the assignment. We construct an independent set as follows: for every triangle we pick a node that corresponds to a satisfied literal (we break ties arbitrarily). It is impossible that two such nodes are adjacent, since only nodes that corresponds to a literal and its negation are adjacent; and they cannot be both satisfied by the assignment.

From Independence to Satisfaction. Suppose we have an independent set I with m vertices. We better have exactly one vertex in I for every triangle. (Two vertices in the same triangle are always adjacent.) Let us fix an assignment that satisfies all the literals that correspond to vertices of I. (Assign values to the other variables arbitrarily.) This is a consistent rule to generate an assignment, because we cannot have a literal and its negation in the independent set). Finally, we note how every clause is satisfied by this assignment.

Wrapping up:

- We showed a reduction $\phi \to (G_\phi, m)$ that given an instance of 3SAT produces an instance of the decision version of Maximum Independent Set.
- We have the property that ϕ is satisfiable (answer YES for the 3SAT problem) if and only if G_ϕ has an independent set of size at least m.
- We knew 3SAT is NP-hard.
- Then also Max Independent Set is NP-hard; and so also NP-complete.

3.2 Maximum Clique

Given a (undirected non-weighted) graph $G = (V, E)$, a *clique* K is a set of vertices $K \subseteq V$ such that *any two* vertices in K are adjacent. In the Maximum Clique problem, given a graph G we want to find a largest clique.

In the decision version, given G and a parameter k, we want to know whether or not G contains a clique of size at least k. It should be clear that the problem is in NP.

We can prove that Maximum Clique is NP-hard by reduction from Maximum Independent Set. Take a graph G and a parameter k, and consider the graph G', such that two vertices in G' are connected by an edge if and only if they are not connected by an edge in G. We can observe that every independent set in G is a clique in G', and every clique in G' is an independent set in G. Therefore, G has an independent set of size at least k if and only if G' has a clique of size at least k.

3.3 Minimum Vertex Cover

Given a (undirected non-weighted) graph $G = (V, E)$, a *vertex cover* C is a set of vertices $C \subseteq V$ such that for every edge $(u, v) \in E$, either $u \in C$ or $v \in C$ (or, possibly, both). In the Minimum Vertex Cover problem, given a graph G we want to find a smallest vertex cover.

In the decision version, given G and a parameter k, we want to know whether or not G contains a vertex cover of size at most k. It should be clear that the problem is in NP.

We can prove that Minimum Vertex Cover is NP-hard by reduction from Maximum Independent Set. The reduction is based on the following observation:

Lemma 4
If I is an independent set in a graph $G = (V, E)$, then the set of vertices $C = V - I$ that are not *in I is a vertex cover in G. Furthermore, if C is a vertex cover in G, then $I = V - C$ is an independent set in G.*

Proof: Suppose C is not a vertex cover: then there is some edge (u, v) neither of whose endpoints is in C. This means both the endpoints are in I and so I is not an independent set, which is a contradiction. For the "furthermore" part, suppose I is not an independent set: then there is some edge $(u, v) \in E$ such that $u \in I$ and $v \in I$, but then we have an edge in E neither of whose endpoints are in C, and so C is not a vertex cover, which is a contradiction. □

Now the reduction is very easy: starting from an instance (G, k) of Maximum Independent set we produce an instance $(G, n-k)$ of Minimum Vertex Cover.

4 Some NP-complete Numerical Problems

4.1 Subset Sum

The **Subset Sum** problem is defined as follows:

- Given a sequence of integers $a_1, \ldots, a_n$ and a parameter k,
- Decide whether there is a subset of the integers whose sum is exactly k. Formally, decide whether there is a subset $I \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in I} a_i = k$.

Subset Sum is a true *decision problem*, not an optimization problem forced to become a decision problem. It is easy to see that Subset Sum is in NP.

We prove that Subset Sum is NP-complete by reduction from Vertex Cover. We have to proceed as follows:

- Start from a graph G and a parameter k.
- Create a sequence of integers and a parameter k'.
- Prove that the graph has vertex cover with k vertices iff there is a subset of the integers that sum to k'.

Let then $G = (V, E)$ be our input graph with n vertices, and let us assume for simplicity that $V = \{1, \ldots, n\}$, and let k be the parameter of the vertex cover problem.

We define integers $a_1, \ldots, a_n$, one for every vertex; and also integers $b_{(i,j)}$, one for every edge $(i, j) \in E$; and finally a parameter k'. We will define the integers a_i and $b_{(i,j)}$ so that if we have a subset of the a_i and the $b_{(i,j)}$ that sums to k', then: the subset of the a_i corresponds to a vertex cover C in the graph; and the subset of the $b_{(i,j)}$ corresponds to the edges in the graph such that exactly one of their endpoints is in C. Furthermore the construction will force C to be of size k.

How do we define the integers in the subset sum instance so that the above properties hold? We represent the integers in a matrix. Each integer is a row, and the row should be seen as the base-4 representation of the integer, with $|E| + 1$ digits.

The first column of the matrix (the "most significant digit" of each integer) is a special one. It contains 1 for the a_is and 0 for the $b_{(i,j)}$s.

Then there is a column (or digit) for every edge. The column (i, j) has a 1 in a_i, a_j and $b_{(i,j)}$, and all 0s elsewhere.

The parameter k' is defined as

$$k' := k \cdot 4^{|E|} + \sum_{j=0}^{|E|-1} 2 \cdot 4^i$$

This completes the description of the reduction. Let us now proceed to analyze it.

From Covers to Subsets Suppose there is a vertex cover C of size k in G. Then we choose all the integers a_i such that $i \in C$ and all the integers $b_{(i,j)}$ such that exactly one of i and j is in C. Then, when we sum these integers, doing the operation in base 4, we have a 2 in all digits except for the most significant one. In the most significant digit, we are summing one $|C| = k$ times. The sum of the integers is thus k'.

From Subsets to Covers Suppose we find a subset $C \subseteq V$ and $E' \subseteq E$ such that

$$\sum_{i \in C} a_i + \sum_{(i,j) \in E'} b_{(i,j)} = k'$$

First note that we never have a carry in the $|E|$ less significant digits: operations are in base 4 and there are at most 3 ones in every column. Since the $b_{(i,j)}$ can contribute at most one 1 in every column, and k' has a 2 in all the $|E|$ less significant digits, it means that for every edge (i, j) C must contain either i or j. So C is a cover. Every a_i is at least $4^{|E|}$, and k' gives a quotient of k when divided by $4^{|E|}$. So C cannot contain more than k elements.

4.2 Partition

The **Partition** problem is defined as follows:

- Given a sequence of integers $a_1, \ldots, a_n$.
- Determine whether there is a partition of the integers into two subsets such the sum of the elements in one subset is equal to the sum of the elements in the other.

  Formally, determine whether there exists $I \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in I} a_i = (\sum_{i=1}^{n} a_i)/2$.

Clearly, Partition is a special case of Subset Sum. We will prove that Partition is is NP-hard by reduction from Subset Sum.[1]

Given an instance of Subset Sum we have to construct an instance of Partition. Let the instance of Subset Sum have items of size $a_1, \ldots, a_n$ and a parameter k, and let $A = \sum_{i=1}^{n} a_i$.

Consider the instance of Partition $a_1, \ldots, a_n, b, c$ where $b = 2A - k$ and $c = A + k$.

Then the total size of the items of the Partition instance is $4A$ and we are looking for the existence of a subset of $a_1, \ldots, a_n, b, c$ that sums to $2A$.

It is easy to prove that the partition exists if and only if there exists $I \subseteq \{1, \ldots, n\}$ such that $\sum_i a_i = k$.

4.3 Bin Packing

The **Bin Packing** problem is one of the most studied optimization problems in Computer Science and Operation Research, possibly the second most studied after TSP. It is defined as follows:

- Given items of size $a_1, \ldots, a_n$, and given unlimited supply of bins of size B, we want to pack the items into the bins so as to use the minimum possible number of bins.

[1]The reduction goes in the non-trivial direction!

You can think of bins/items as being CDs and MP3 files; breaks and commercials; bandwidth and packets, and so on.

The decision version of the problem is:

- Given items of size $a_1, \ldots, a_n$, given bin size B, and parameter k,
- Determine whether it is possible to pack all the items in k bins of size B.

Clearly the problem is in NP. We prove that it is NP-hard by reduction from Partition.

Given items of size $a_1, \ldots, a_n$, make an instance of Bin Packing with items of the same size and bins of size $(\sum_i a_i)/2$. Let $k = 2$.

There is a solution for Bin Packing that uses 2 bins if and only if there is a solution for the Partition problem.