Lecture 8: NP-complete programs

Anup Rao

January 26, 2023

The concept of NP-completeness makes sense because we do know of examples of **NP**-complete problems.

Circuit-Sat

Definition 1. *CircuitSat* : $\{0,1\}^* \to \{0,1\}$ *is the function that views its input as a circuit* C *and outputs* 1 *iff* $\exists x$ *such that* $C(x) = 1$.

I have claimed in class that circuits can simulate Turing Machines. Here is what you can actually prove in this regard:

Theorem 2. *If a function* $f : \{0,1\}^* \to \{0,1\}$ *can be computed in time* $t(n)$ *by a Turing machine, then for every* n *there is a circuit of size* $O(t(n) \log t(n))$ *that computes* f *restricted to the inputs of size* n.

Although we did not prove this theorem in class, we sketched how you could find a circuit of size $O(t(n)^2)$ that computes f. The idea was to add a layer of gates that maintains the entire state of the Turing machine—contents of all tapes, pointers, and the line of code being executed. Then we add a new layer that computes this configuration after one execution step of the Turing machine, using the earlier configuration as input. A single configuration can be written down using $O(t(n))$ gates since we only need to write down the values of the tapes up to $O(t(n))$ coordinates. The new configuration can be computed from the old one with $O(t(n))$ gates as well. After repeating this $O(t(n))$ times, we obtain the final configuration of the Turing machine, which must include the value of $f(x)$.

Theorem 3. *CircuitSat is* **NP***-complete.*

Proof It is clear that *CircuitSat* is in **NP**. Next we show that for every $f \in \mathbf{NP}$, $f \leq_P$ *CircuitSat*. Let V be a verifier for f. Then to compute $f(x)$, the reduction will build the circuit $C_x(w)$ that computes $V(x,w)$, where here w are the input variables to the circuit and x is the input. Since $f(x) = 1$ if and only if there exists w such that $C_x(w) = 1$, we can determine the value of f by computing *CircuitSat*(C_x). ■

3SAT

A *boolean formula* is an expression of the form

$$(x_1 \wedge \neg x_2) \vee (x_7 \wedge \neg(x_6 \vee \neg x_2)).$$

Formally: it is a circuit where the only allowed gates are $\vee, \wedge, \neg$, and every gate has fan-out at most 1. Input gates are allowed to repeat. As usual, size of the gates is number of gates, and the fan-in is allowed to be at most 2. The formula is said to be in conjunctive normal form (CNF) if it is an AND of OR's. Similarly, it is said to be in disjunctive normal form (DNF) if it is an OR of ANDS. For example

$$(x_1 \vee \neg x_2) \wedge (\neg x_7 \vee x_9 \vee \neg x_1)$$

is a CNF.

We have the following lemma:

Lemma 4. *Every function $f : \{0,1\}^\ell \to \{0,1\}$ can be computed by a CNF (resp. DNF) of size $\ell 2^\ell$.*

Proof For each input z such that $f(z) = 0$, we add the literal x_i to the clause if $z_i = 0$ and $\neg z_i$ otherwise. So for example, if $f(0,1,0) = 0$, we add the clause $(x_1 \vee \neg x_2 \vee x_3)$. Then note that each clause is 0 on exactly one input, and all inputs x for which $f(x) = 0$ make some clause 0. Every other input evaluates to 1. So, the CNF computes f. The resulting formula is of size $\ell 2^\ell$. The case of DNF's is symmetric. ■

We define $\mathsf{SAT} : \{0,1\}^* \to \{0,1\}$ to be the function that takes as input a boolean formula F, and outputs 1 if and only if there is a an x such that $F(x) = 1$. A 3-CNF formula is a CNF where every clause has at most 3 variables. For example:

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_3 \vee x_4 \vee \neg x_1) \wedge \cdots .$$

$\mathsf{3SAT} : \{0,1\}^* \to \{0,1\}$ is the function that takes as input 3-CNF and outputs 1 if and only if the formula is satisfiable. Next we show that even this function is **NP**-complete

Theorem 5. 3SAT *is* **NP***-complete.*

Proof $\mathsf{3SAT} \in \mathbf{NP}$ is easy enough to check. The witness is a satisfying assignment to the formula. The verifier simply evaluates the formula on the given witness, and outputs the results of the evaluation.

Since we have already shown that $\mathsf{CKT-SAT}$ is **NP**-hard, it will be enough to show that $\mathsf{CKT-SAT} \leq_P \mathsf{SAT}$.

Is the same true for 2SAT? We do not know. There are polynomial time algorithms for 2SAT, so if you found a reduction to 2SAT, you would prove $\mathbf{P} = \mathbf{NP}$. The algorithm works by viewing every clause $(x \vee y)$ as an implication $\neg x \Rightarrow y$ as well as the implication $\neg y \Rightarrow x$. This defines a directed graph where all the vertices correspond to variables and their negations, and the edges correspond to implications. You can show that the formula is satisfiable if and only if there is no path that leads from a variable to its negation.

Given a circuit, we shall output a CNF formula that is satisfiable if and only if the circuit accepts some input. Introduce a new variable y_g for each internal gate g of the circuit. If the internal gate g has inputs h, q, let F_g be the CNF formula on variables y_g, y_h, y_q that is 1 if and only if $y_g = g(y_q, y_h)$. By Lemma 4, this formula is a 3-CNF of constant size. If the output gate is v, the final formula is

$$y_v \wedge \bigwedge_g F_g,$$

which is satisfied if and only if the circuit has a satisfying assignment.

Every clause of this formula has at most 3 variables. To make sure it has *exactly* 3 variables, we replace each clause with less than 3 variables with a 3-CNF that by adding dummy variables. For example, we can replace y_v by a 3-CNF on the variables y_v, z_1, z_2 that computes the same function as y_v:

$$(y_v \vee z_1 \vee z_2) \wedge (y_v \vee \neg z_1 \vee z_2) \wedge (y_v \vee \neg z_1 \vee \neg z_2) \wedge (y_v \vee z_1 \vee \neg z_2).$$

■

We now consider several interesting graph problems and show that they are **NP**-complete.

3 Coloring

We say that an undirected graph is 3 colorable if one can color every vertex with one of 3 colors so that every edge gets two colors.

$$\mathsf{3COL}(G) = \begin{cases} 1 & \text{if } G \text{ is 3 colorable} \\ 0 & \text{otherwise.} \end{cases}$$

Although there is an easy polynomial time algorithm for 2-coloring a graph (greedily color the first vertex blue, all its neighbors red, all their neighbors blue and so on), we know of no such algorithm for 3-coloring a a graph.

Theorem 6. 3COL *is* **NP***-complete.*

Sketch of Proof The coloring serves as a witness that can be verified in polynomial time, so $\mathsf{3COL} \in \mathbf{NP}$. Next we show how to reduce 3SAT to 3COL in polynomial time.

We would like to construct the graph in a way that allows every coloring to be decoded to an assignment to the variables. To this end,

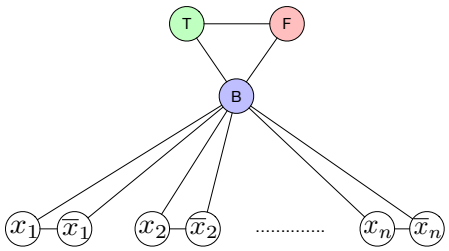

Figure 1: Ensuring that a coloring corresponds to a truth assignment

we shall have three vertices named T, F, B and $2n$ vertices named $x_1, \overline{x}_1, x_2, \overline{x}_2, \ldots, x_n, \overline{x}_n$ that correspond to the variables and their negations. We shall connect every pair of T, F, B so that these three must be given a distinct color. We also connect each x_i and $\overline{x}_i$ to B, so x_i and $\overline{x}_i$ must be given the same as color as T or F. In addition, connect each x_i and $\overline{x}_i$ to ensure that they are assigned the same color. (See Figure 1). Thus any coloring corresponds to an assignment of truth values to the variables.

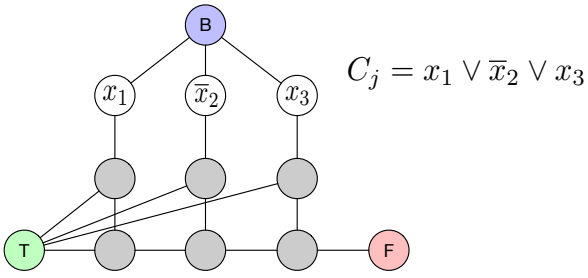

Figure 2: Ensuring that the assignment satisfies each clause

My initial drawing in class had an error!

Next we need to encode each clause of the formula. The idea here is generate a part of the graph that can be colored if and only if the clause is satisfied by the assignment to the corresponding variables. This is shown in Figure 2. We connect the gadget shown there to the variables that correspond to the clause we are interested in. If any one of the variables is set to T, then one can color the corresponding vertex in the top row F. This allows us to color the bottom row.

On the other hand, if all variables in the clause are false, then the top row must be colored B, and the bottom row cannot be colored correctly.

■

Independent Set

Given an undirected graph G, an *independent set* in the graph is a set of vertices such that no edge is contained in the set. The goal is find an independent set of maximum size in the graph. We can encode this problem using the following boolean function:

$$\mathsf{ISET}(G,k) = \begin{cases} 1 & \text{if } G \text{ has an independent set of size } k, \\ 0 & \text{otherwise.} \end{cases}$$

If you can compute ISET in polynomial time, then you can find the largest independent set in polynomial time (how?). If on the other hand you can find the largest independent set, then you can also compute ISET. Here we prove:

Theorem 7. ISET *is* **NP***-complete.*

Proof ISET is in **NP**, since the independent set of largest size is itself a witness which can be verified in polynomial time. Thus it only remains to show that ISET is **NP**-hard. To do this, we show how to reduce 3SAT to ISET in polynomial time.

Given a 3SAT instance with m clauses and n variables, we construct a graph with $3m$ variables. Each clause C_i corresponds to 3 vertices, which are all connected to each other. Thus the graph contains m disjoint triangles. In each triangle, we label each of the three vertices with the three literals that occur in the clause. Thus the clause $(a \vee \neg b \vee c)$ leads to the three vertices being labeled $a, \neg b, c$. Finally, for every variable a, we connect every vertex labeled a to every vertex labeled $\neg a$ using an edge.

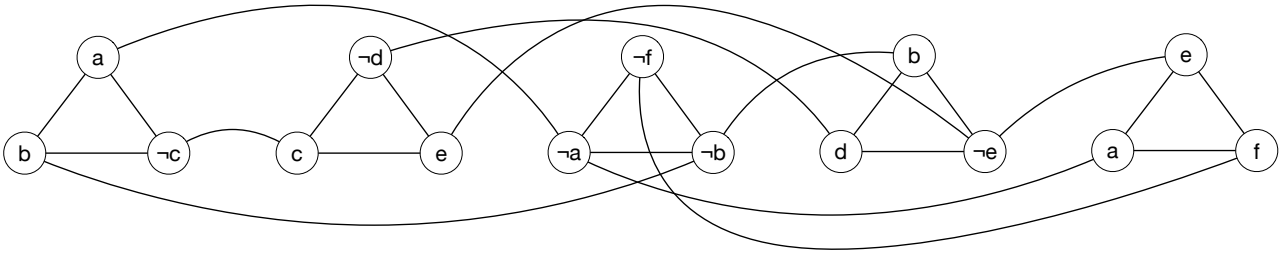

Figure 3: An example of the input to ISET produced when the input formula is $(a \vee b \vee \neg c) \wedge (\neg d \vee c \vee e) \wedge (\neg f \vee \neg a \vee \neg b) \wedge (b \vee d \vee \neg e) \wedge (e \vee a \vee f)$.

We claim that the above graph has an independent set of size m if and only if the given 3 CNF is satisfiable. Indeed, suppose the 3 CNF is satisfiable using the assignment to the variables x. Then x must satisfy every clause, so in each clause, some literal must be true. Pick a single vertex from each of the triangles in such a way that we always pick a true vertex. By the construction, every edge

must connect a true vertex to a false vertex, so the resulting set is independent. There cannot be a larger independent set in the graph, since every triangle can contain only one vertex.

Conversely, if the graph has an independent set of size m, then there must be exactly one vertex in every triangle of the construction, or else one of the triangle edges would be included in the set. Now pick the assignment to the variables in such a way that all the vertices of the independent set are labeled with true. There is always a way to do this, since by construction every time we try to set a variable in this process, it has not already been set to a different value by the construction of the graph and the property that the set is independent.

Thus the reduction is to read the input formula and construct the above graph in polynomial time. ■