Lecture 7: NP-Complete Problems

Anup Rao

April 20, 2021

In the previous class we defined the notion of **NP**-completeness and showed that *CircuitSat* is **NP**-complete. In this lecture, we will consider few other natural problems and prove that they are **NP**-complete. We first discuss the **NP**-completeness of 3SAT.

At the beginning of lecture, I tried to show another **NP**-complete problem that would simplify the presentation of **NP**-completeness. I did not include this in the notes, because I don't think it actually made anything easier to understand!

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 1. *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 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 2. $\mathsf{3SAT}$ *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 $\mathsf{CKT} - \mathsf{SAT}$ is **NP**-hard, it will be enough to show that $\mathsf{CKT} - \mathsf{SAT} \leq_P \mathsf{SAT}$.

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 1, 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.

Is the same true for $\mathsf{2SAT}$? We do not know. There are polynomial time algorithms for $\mathsf{2SAT}$, so if you found a reduction to $\mathsf{2SAT}$, 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.

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

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

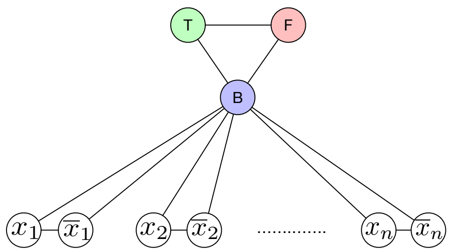

Figure 1: Ensuring that a coloring corresponds to a truth assignment

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, 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.

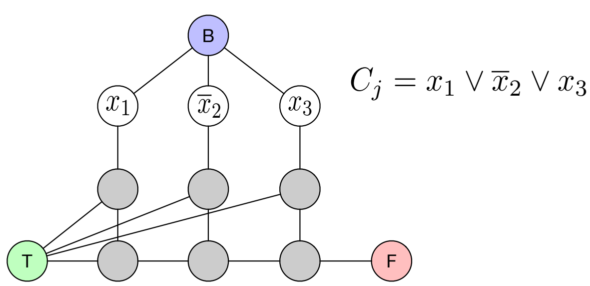

Figure 2: Ensuring that the assignment satisfies each clause

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 4. 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.

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

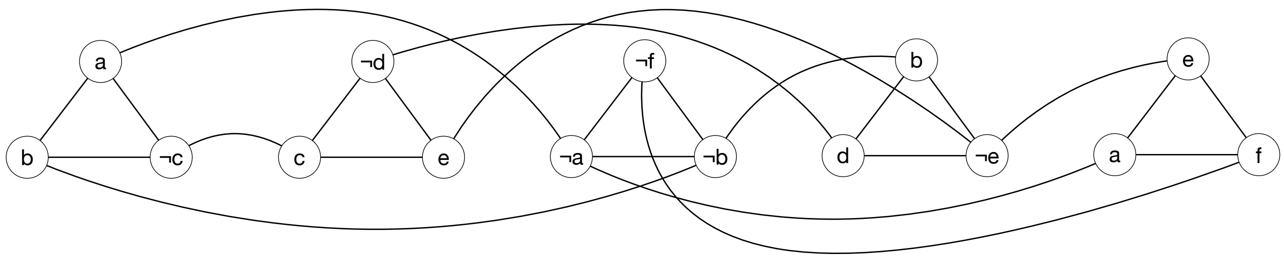

Figure 3: An example of the input to ISET produced when the input formula is $(a \vee b \vee \neg c) \wedge (\neg d \vee c \vee e) \wedge (\neg f \vee \neg a \vee \neg b) \wedge (b \vee d \vee \neg e) \wedge (e \vee a \vee f)$.

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. ■

Hamiltonian Path

Given a directed graph G, a Hamiltonian path is a path that visits every vertex of the graph exactly once. We define the function

$$\mathsf{HPATH}(G) = \begin{cases} 1 & \text{if } G \text{ has a Hamiltonian path} \\ 0 & \text{otherwise.} \end{cases}$$

Theorem 5. HPATH *is* **NP***-complete.*

Proof Given a path in the graph, one can check in polynomial time whether or not it is a Hamiltonian path. Thus HPATH $\in$ **NP** using the path as a witness. Next we show that you can reduce 3SAT to HPATH, proving that HPATH is **NP**-hard.

Suppose the formula has n variables and m clauses. We shall construct a graph on $(2m+2)n+2$ vertices that encodes assignments to the formulas as follows. We start by constructing a graph that will contain $(2m+2)n$ vertices named $v_{i,j}$, where $i \in \{1, 2, \ldots, n\}$ and $j \in$. For every i and $1 \le j < j+1 \le k$, we have the edges $(v_{i,j}, v_{i,j+1})$ and $(v_{i,j+1}, v_{i,j})$. Thus these vertices can be thought of as arranged

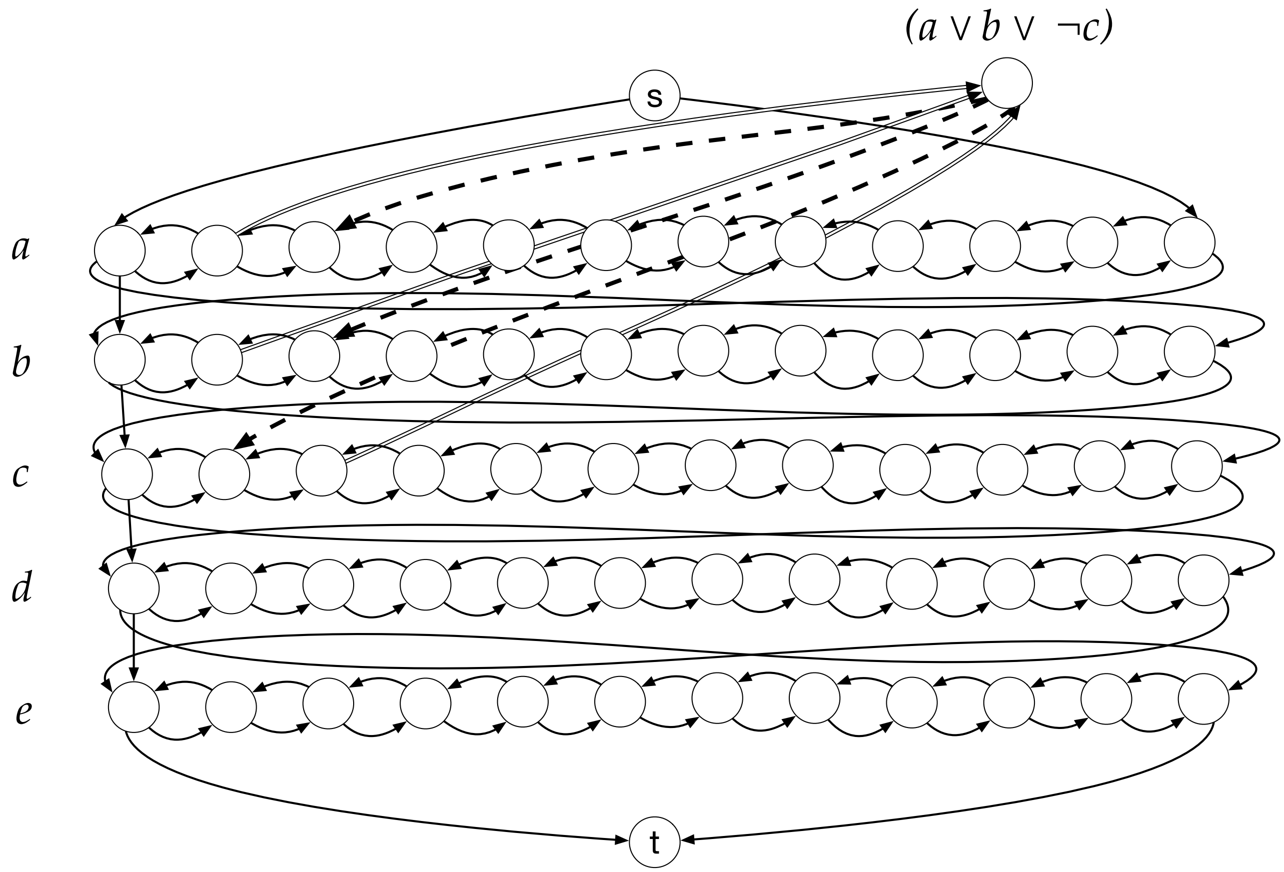

Figure 4: An example showing how to generate a directed graph for the Hamiltonian path problem using a single clause from the formula.

in n rows, where in each row the path can go left or right. For every $1 \leq i < i+1 \leq n$, we add the edges

$$(v_{i,1}, v_{i+1,1}), (v_{i,1}, v_{i+1,n}), (v_{i,n}, v_{i+1,1}), (v_{i,n}, v_{i+1,1}).$$

Finally we add two special vertices s, t, with edges

$$(s, v_{1,1}), (s, v_{1,n}), (v_{n,1}, t), (v_{n,n}, t).$$

By construction, every Hamiltonian path in the graph must start at s and end at t, and must traverse each row in order. Each row can be traversed in either left to right or right to left fashion. We shall imagine that traversing the row left to right corresponds to assigning the i'th variable the value 0, and traversing it the other way corresponds to assigning the value 1.

Next we add some vertices to encode the constraints given by the clauses. Without loss of generality we assume that each clause contains a variable at most once (since we can always reduce the formula to this case). For the j'th clause C_j, we add the vertex c_j. For every variable x_i that the clause contains unnegated, we add the edges $(v_{i,2j}, c_j), (c_j, v_{i,2j-1})$. For every variable x_j that is contained in the clause as $\neg x_j$, we add the edges $(v_{i,2j-1}, c_j), (c_j, v_{i,2j})$. By construction, any Hamiltonian path that takes the edge $(v_{i,2j}, c_j)$, must take $(c_j, v_{i,2j-1})$ next, or $v_{i,2j-1}$ will never be visited. Similarly, any Hamiltonian path that takes the edge $(v_{i,2j}, c_j)$ must take $(c_j, v_{i,2j-1})$ next. We claim that the graph has a Hamiltonian path if and only if the formula is satisfiable.

Indeed, if the formula is satisfiable, then traverse each row in the direction corresponding to the satisfying assignment. Since each clause is satisfied by some variable, we can visit the vertex for the clause when we traverse the first variable that satisfies it. Conversely, if there is a Hamiltonian path, then the construction ensures that this path corresponds to an assignment to the variables, and this path must visit every clause vertex, which guarantees that each clause vertex is satisfied by some variable. ■

Subset Sum

In the subset sum problem, the input is a collection of numbers $a_1, \ldots, a_k$, as well as a target number t. The goal is compute whether or not some subset of the numbers $a_1, \ldots, a_k$ sums to t.

$$\mathsf{SubSum}(a_1, \ldots, a_k, t) = \begin{cases} 1 & \text{if there is a subset } S \subseteq \{1, 2, \ldots, n\} \text{ such that } \sum_{i \in S} a_i = t, \\ 0 & \text{otherwise.} \end{cases}$$

Theorem 6. SubSum *is* **NP**-*complete.*

We sketch the proof. SubSum is in **NP**, since there is an obvious polynomial time computable verifier for the problem. The witness is a subset S, and the verifier simply checks that $\sum_{i \in S} a_i = t$, which can be done in polynomial time.

To show that SubSum is **NP**-hard, we shall show that

$$\mathsf{3SAT} \leq_P \mathsf{SubSum}.$$

We describe the polynomial time reduction next. Given a 3-sat formula ϕ, our algorithm needs to output numbers $a_1, \ldots, a_k$ and t such that $\mathsf{SubSum}(a_1, \ldots, a_k, t) = 1$ if and only if ϕ is satisfiable.

Suppose ϕ has n variables and m clauses. Then, we will have $k = 2n + 2m$, and all of the numbers $a_1, \ldots, a_k$ and t will be $n + m$ digit numbers, written in base 10. Moreover, all the digits of $a_1, \ldots, a_k$ will be either 0 or 1, and the numbers will be chosen in such a way that adding any subset of $a_1, \ldots, a_k$ will never produce a carry.

For each variable x_i of the formula ϕ, we shall have two numbers: t_i and f_i. The i'th digit of t_i and f_i will be set to 1 and all of the remaining $n-1$ digits in the first n digits will be set to 0. Meanwhile, in the target number t, all of the first n digits will be set to 1. This choice ensures that choosing any subset of $t_1, f_1, \ldots, t_n, f_n$ that sums to t corresponds to choosing either t_i or f_i to be included in the set, for each i. In other words, a subset of these numbers that sums to t corresponds to a truth assignment to the variables $x_1, \ldots, x_n$. Next, we need to add more digits to ensure that this truth assignment satisfies all the clauses. For every i, j, if x_i occurs in the j'th clause, we make the $n+j$'th digit of t_i 1. If $\neg x_i$ occurs in the j'th clause, we make the $n+j$'th digit of f_i 1. All other digits (upto the $n+m$'th digit) of t_i, f_i are set to 0. This choice ensures that if the subset chosen satisfies the j'th clause, then the j'th digit of the sum will be either $1, 2$ or 3. Finally, we add two numbers b_j, c_j, which are 0 in all digits, except for the j'th digit. The j'th digit of both numbers is 1. This ensures that if the j'th clause is satisfied by the assignment, then one can pick $0, 1$ or 2 elements of $\{b_j, c_j\}$ to add to the subset, so that the sum of the j'th digits is 3.

Example: suppose we are given the formula $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_2 \vee x_3 \vee x_4) \wedge (\neg x_1, \vee \neg x_3 \vee \neg x_4) \vee (\neg x_2, \vee \neg x_3 \vee x_4)$. There are 4 variabels and 4 clauses, so the polynomial time reduction will generate 16 numbers, each with 8-digits, and a target number with 8-digits:

$$\begin{aligned}
t_1 &= 10001000\\
f_1 &= 10000010\\
t_2 &= 01000000\\
f_2 &= 01001101\\
t_3 &= 00101100\\
f_3 &= 00100011\\
t_4 &= 00010101\\
f_4 &= 00010010\\
b_1 &= 00001000\\
c_1 &= 00001000\\
b_2 &= 00000100\\
c_2 &= 00000100\\
b_3 &= 00000010\\
c_3 &= 00000010\\
b_4 &= 00000001\\
c_4 &= 00000001
\end{aligned}$$

The target number will be:

$$t = 11113333.$$