Lecture 8: More NP-Complete Problems

Anup Rao

October 23, 2018

We continue our discussion of **NP**-complete problems. The main take-away of these results is that a wide variety of different kinds of problems turn out to be **NP**-complete.

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 1. 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

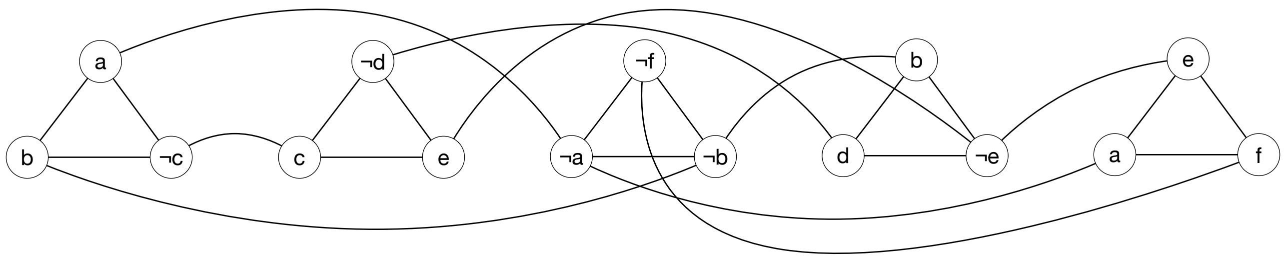

Figure 1: An example of the input to ISET produced when the input formula is $(a \vee b \vee \neg c) \wedge (\neg d \vee c \vee e) \wedge (\neg f \vee \neg a \vee \neg b) \wedge (b \vee d \vee \neg e) \wedge (e \vee a \vee f)$.

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. ∎

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 2. 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 $1 \leq j < j+1 \leq k$, we have the edges $(v_{i,j}, v_{i,j+1})$

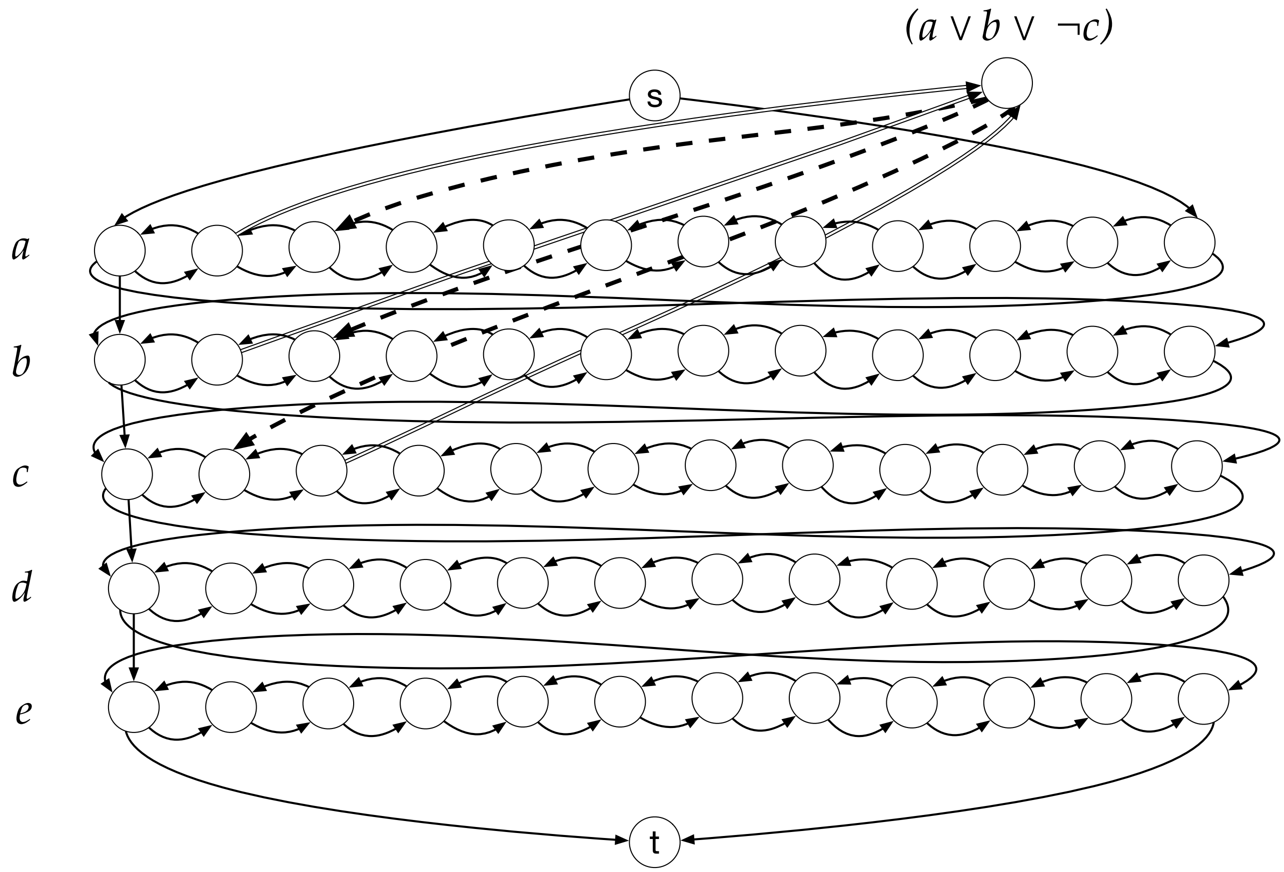

Figure 2: An example showing how to generate a directed graph for the Hamiltonian path problem using a single clause from the formula.

and $(v_{i,j+1}, v_{i,j})$. Thus these vertices can be thought of as arranged 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 $a_1, \dots, a_k$, as well as a target number t. The goal is compute whether or not some subset of the numbers $a_1, \dots, a_k$ sums to t.

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

Theorem 3. 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 $a_1, \dots, a_k$ and t such that $\mathsf{SubSum}(a_1, \dots, a_k, t) = 1$ 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 $a_1, \dots, a_k$ and t will be $n + m$ digit numbers, written in base 10. Moreover, all the digits of $a_1, \dots, a_k$ 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:

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

The target number will be:

$$t = 11113333.$$