

Lecture 20: November 21

Lecturer: Micah Adler

Scribes: Georgios Papadopoulos

In this lecture:

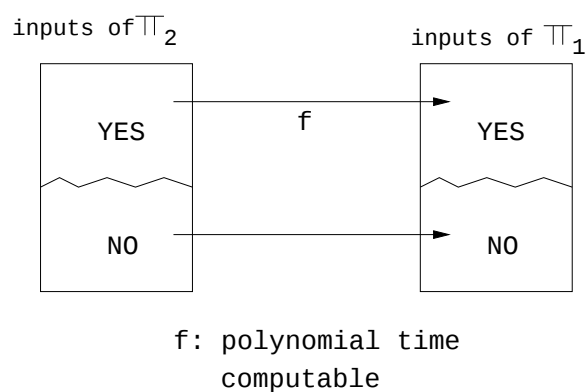
- NP-completeness
- Clique
- Vertex Cover

20.1 Review

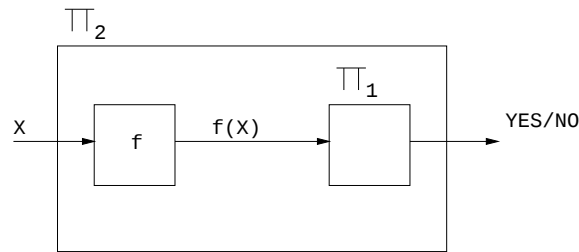
20.1.1 Polynomial-Time Reduction

Definition 20.1 π_2 is polynomial-time reducible to π_1 , ($\pi_2 \leq_p \pi_1$), if and only if there exists a polynomial-time algorithm “f” that transforms any instance x of π_2 to an instance $f(x)$ of π_1 such that,

x is a YES instance of $\pi_2 \Leftrightarrow f(x)$ is a YES instance of π_1 , and
 x is a NO instance of $\pi_2 \Leftrightarrow f(x)$ is a NO instance of π_1 .

Figure 20.1: Polynomial time reduction from π_2 to π_1 .

Lemma 20.2 If $\pi_2 \leq_p \pi_1$ and if $\pi_1 \in \mathbf{P}$, then $\pi_2 \in \mathbf{P}$. If $\pi_2 \leq_p \pi_1$ and if $\pi_2 \notin \mathbf{P}$, then $\pi_1 \notin \mathbf{P}$.

Figure 20.2: Polynomial-time reduction algorithm for π_2 .

Let function f be a polynomial-time reduction algorithm that computes $f(x)$ for each input instance x of π_2 . For a given input $x \in \pi_2$ the function f transforms x into $f(x)$ and this can be used as input to π_1 which outputs *YES* or *NO* answers for the original problem π_2 . The transformation takes polynomial-time and so does solving problem π_1 as $\pi_1 \in \mathbf{P}$. So the algorithm for solving problem π_2 runs in polynomial time. The correctness of π_2 follows from the above definition.

Definition 20.3 *Decision problem π is **NP-Complete** if and only if,*

1. $\pi \in \mathbf{NP}$, and
2. $\forall \pi' \in \mathbf{NP}, \pi' \leq_p \pi$.

20.1.2 NP-complete problems

To show that problem π is **NP-complete**, we need to

- show that $\pi \in \mathbf{NP}$, and
- find a π^* that is **NP-complete** and show that $\pi^* \leq_p \pi$.

20.1.3 3-SAT

The 3-SAT problem [CLR90] is **NP-complete** and this can be used to prove that other problems are **NP-complete**.

Definition 20.4 *The 3-SAT problem is defined as follows:*

Input: A boolean formula ϕ in 3-CNF (Conjunctive Normal Form).

Question: Is ϕ satisfiable? (i.e. Is there some assignment of values to variables that makes the value of ϕ true?)

Notation: $\phi = (l_{11} \vee l_{12} \vee l_{13}) \wedge (l_{21} \vee l_{22} \vee l_{23}) \dots$

20.2 The Clique problem

Definition 20.5 A clique of size k in graph G is a completely connected subgraph of G with k vertices.

Figure 3 shows an example.

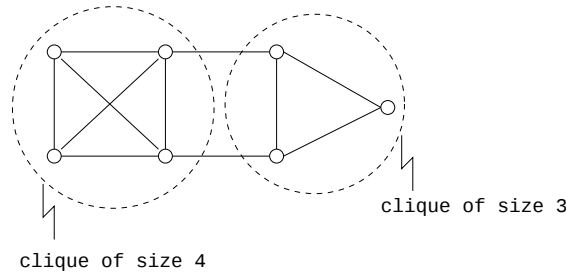


Figure 20.3: Sample graph with two cliques.

Definition 20.6 The clique decision problem is:

Input: Graph $G(V, E)$ and integer k .

Question: Does G contain a clique of size $\geq k$?

Theorem 20.7 Clique is **NP**-complete.

Proof:

1. First prove that Clique $\in$ **NP**.

Using the definition of **NP**, algorithm $A(G, y)$ verifies that y is a clique over the graph described by $G = (V, E)$. The verification can be done in polynomial time by checking that $y \subseteq V$, that $\forall u, v \in y \langle u, v \rangle \in E$, and that $|y| \geq k$.

2. Next prove that 3-SAT $\leq_p$ Clique.

We need to find a *polynomial-time reduction* that converts an input instance x of 3-SAT to a corresponding input instance of the clique problem.

$$\text{CNF formula } \phi \xrightarrow[\text{Function } f]{\text{polynomial time transformation}} G_\phi, k_\phi$$

Properties of the transformation:

- (a) It must be polynomial time.
- (b) If G_ϕ has a clique of size k_ϕ , then ϕ must be satisfiable.
- (c) If ϕ is satisfiable then G_ϕ must have a clique of size k_ϕ .

G_ϕ has a clique of size k_ϕ if and only if ϕ is satisfiable, i.e. “YES” instances of the 3-SAT problem are mapped to “YES” instances of the clique problem and “NO” instances of 3-SAT are mapped to “NO” instances of clique.

Proposed transformation:

k_ϕ = the number of clauses in ϕ .

$G_\phi = (E, V)$ such that:

Node $n_{ij} \in V \iff l_{ij} \in \phi$. The nodes of the graph correspond to the literals in the formula.

Edge $\langle n_{ij}, n_{hk} \rangle \in E$ if $i \neq h$ (the literals belong to different clauses) and $l_{ij} \neq \neg l_{hk}$, both literals can consistently be set to *true* at the same time.

Example 20.8

$$\phi = (a \vee \bar{b} \vee c) \wedge (\bar{a} \vee b \vee \bar{c}) \wedge (\bar{a} \vee \bar{b} \vee d)$$

Thus: $k_\phi = 3$

G_ϕ :

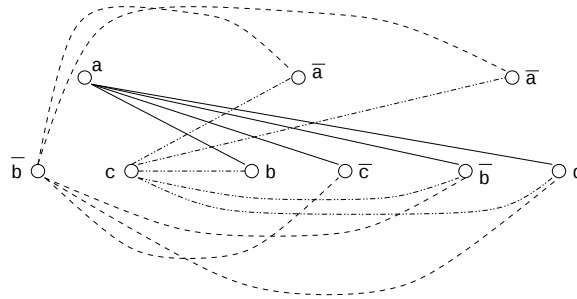


Figure 20.4: Graph G_ϕ derived from the example 3-CNF formula ϕ . Edges are shown only for the first clause of ϕ .

We need to show that the properties of transformation are satisfied.

- (a) The graph G_ϕ can be computed from ϕ in polynomial time. To compute k_ϕ , we need to count the number of clauses. To create nodes of graph assign one node for each literal. To create edges, consider each possible node pair and check whether there exists an edge between the two nodes.
- (b) **Claim 20.9** If G_ϕ has a clique of size k_ϕ , then ϕ is satisfiable.

No edges in G_ϕ connect vertices in the same clause and so each vertex in the clique corresponds to exactly one literal per clause. Also since no edges are present between a node for a literal and a node for the complement of the literal we can safely set all the literals that correspond to nodes in the clique to *true*. Each clause is thus *true* and so ϕ is satisfied.

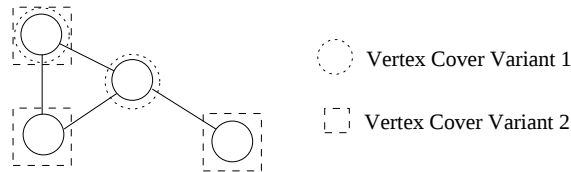
- (c) **Claim 20.10** If ϕ has a satisfying assignment, then G_ϕ has a clique of size k_ϕ .

At least one literal has to be *true* in each clause under the satisfying assignment. The literals are consistent as they cannot be *true* and *false* at same time, i.e. $x = \text{true}$ and $\bar{x} = \text{true}$ is not possible by the nature of construction of G_ϕ . Pick one *true* literal per clause and we form a corresponding set of nodes for those literals in G_ϕ . As no node is a complement of any other node, we have edges from each node to all other nodes in the set. This gives us a clique of size k .

Thus, we have reduced the 3-SAT problem in polynomial time to the clique problem and the transformation used satisfies all the required properties. This concludes the proof of the theorem that clique is **NP**-complete. ■

20.2.1 Vertex Cover

Definition 20.11 A vertex cover of the undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ of the vertices of G such that the vertices in V' touch every edge in E , i.e. $\forall e \in E : \exists v \in V', w \in V : e = (v, w)$.



Example 20.12 Figure 20.5: A graph and two possible vertex covers.

Definition 20.13 The vertex cover problem.

Input: A graph $G = (V, E)$, and an integer k .

Question: Does G contain a vertex cover V' of size $\leq k$?

Theorem 20.14 The vertex cover problem is **NP**-complete.

Proof:

1. The vertex cover problem $\in$ **NP**.

We can verify that a witness $W \subseteq V$ is a vertex cover of G in polynomial time by inspecting each edge $e \in E$ to verify that it is touched by a vertex in W .

2. $3SAT \leq_p VERTEX - COVER$.

Idea: Solve the satisfiability problem for any 3 - *CNF* formula Φ by transforming Φ into (G_ϕ, k_ϕ) and solving with (G_ϕ, k_ϕ) as input the corresponding vertex cover problem.

The proposed transformation:

Set $k_\phi = V + 2m$, where V is the number of variables and m is the number of clauses.

Build G_ϕ as follows:

For each variable X add a subgraph that looks like this:

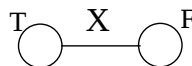


Figure 20.6: The subgraph for a variable.

For each clause $(l_1 \vee l_2 \vee l_3)$ add a subgraph that looks like this:

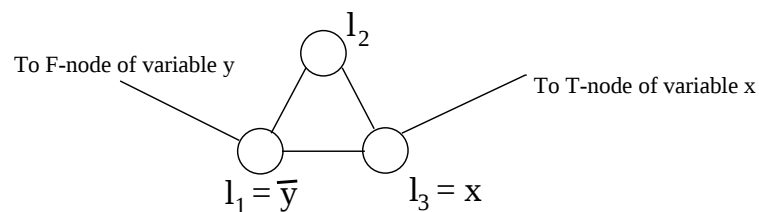


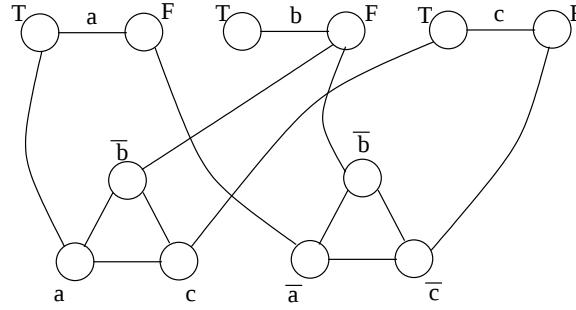
Figure 20.7: The subgraph for a clause.

The transformation of the formula ϕ into G_ϕ, k_ϕ can be performed in polynomial time since to construct G_ϕ only V variables, m clauses and the $3*m$ literals must be considered to create and connect the corresponding nodes of G_ϕ .

Example 20.15 $\Phi = (a \vee \bar{b} \vee c) \wedge (\bar{a} \vee \bar{b} \vee \bar{c})$

$$k_\phi = V + 2m = 3 + 2 * 2 = 7$$

G_ϕ :

Figure 20.8: The graph corresponding to $(a \vee \bar{b} \vee c) \wedge (\bar{a} \vee \bar{b} \vee \bar{c})$.

Claim 20.16 If Φ is satisfiable, then G_ϕ has a vertex cover of size $V + 2m$.

Proof: If Φ is satisfiable, there must exist an assignment $A = \{x_1 = \text{True}, x_2 = \text{False}, \dots\}$ of the variables that makes $\Phi(A)$ true. The subset W of the vertices of G_Φ is now chosen as follows:

For each variable x in Φ , we have two vertices x_T, x_F in G_Φ . If $x = \text{true}$ in the assignment A , include x_T in W otherwise x_F .

After having done this for each variable W contains V vertices. These vertices cover the segments of G_Φ that correspond to variables in Φ and all the edges that connect nodes in the “variable” and “clause” components of G_Φ that correspond to true literals under the assignment A in $\Phi(A)$.

For each clause (l_1, l_2, l_3) in Φ , we have three vertices v_{l_1}, v_{l_2} , and v_{l_3} . Since the assignment A makes $\Phi(A)$ true, there must be at least one literal l_i in each clause which is true. The corresponding vertex v_{l_i} does not need to be included in the vertex cover W , because the edge from v_{l_i} to the corresponding “variable” vertex x_T (or x_F , if the $l_i = \bar{x}$) is already covered by x_T (or x_F) which is in W . So we include the remaining vertices of the clause v_{l_j} and v_{l_k} in W . Whereby all the edges in the “clause” part of G_Φ are covered and also the yet untouched edges connecting the “variable” and “clause” components of the graph. Thus W is a vertex cover which contains $V + 2m$ vertices and the above claim is proven. ■

Claim 20.17 If G_Φ has a vertex cover of size $V + 2m$, then Φ is satisfiable.

Proof: Suppose that a vertex cover W of size $V + 2m$ on G_ϕ is given. Looking at some variable x in Φ and its corresponding vertices x_T and x_F in G_Φ , we can see that one of these vertices x_T, x_F must be in W , otherwise the edge connecting the two vertices would not be touched by any vertex in W . That means W has to contain at least V “variable” vertices and each variable contributes at least one “variable” vertex to

W . Looking at some clause (l_1, l_2, l_3) in Φ , and its corresponding vertices v_{l_1} , v_{l_2} , and v_{l_3} in G_Φ , we can see that at least two of these vertices must be in W , otherwise one edge of this “clause” segment of the graph would not be touched by any vertex in W . So W has to contain at least $2m$ “clause” vertices and each clause contributes at least two “clause” vertices to W .

Since W contains only $V + 2m$ vertices it follows that each variable contributes exactly one “variable” vertex and each clause contributes exactly two “clause” vertices to W . We can choose an assignment A such that for each variable x in Φ , if $x_T \in W$, then $x = \text{true}$ in A else $x = \text{False}$.

We can show via proof by contradiction that $\Phi(A)$ is *true*.

Assume that $\Phi(A)$ is *false*. If $\Phi(A)$ is *false*, one clause c of Φ must be false under the assignment A . This means that all the literals in c are *false* under the assignment A . In the clause c , we have the “clause” vertices v_{l_1} , v_{l_2} , and v_{l_3} . There are only two “clause” vertices say v_{l_1} and v_{l_2} in W . So the edge e coming out of v_{l_3} must be covered by the “variable” vertex at the other end of e . Let us now consider two cases.

Case 1: Let the corresponding literal l_3 to v_{l_3} be of the form x , then at the other end of e the “variable” vertex is x_T . Following the above argument x_T must be in the vertex cover W (otherwise e would not be touched by any vertex). But if this is true, the previously constructed assignment A has set the corresponding variable $x = \text{true}$. Thus the literal $l_3 = x$ is *true* under the assignment A . This contradicts the fact that all the literals in c including l_3 were assumed to be *false* under the assignment A .

Case 2: Let corresponding literal l_3 to v_{l_3} be of the form $\bar{x}$, then at the other end of e the “variable” vertex is x_F . Following the above argument x_F must be in the vertex cover W (otherwise e would not be touched by any vertex). But if this is true, the previously constructed assignment A has set the corresponding variable $x = \text{False}$. Thus the literal $l_3 = \bar{x}$ is *true* under the assignment A . This contradicts the fact that all the literals in c including l_3 were assumed to be *false* under the assignment A .

Both possible cases yield a contradiction thus $\Phi(A)$ is *true* and therefore Φ is satisfiable and the above claim is proven. ■

The last two proven claims conclude the proof of VERTEX-COVER being **NP**-complete. ■

20.3 Reference

CLR90 THOMAS H. CORMEN, CHARLES E. LEISERSON, and RONALD L. RIVEST, Introduction to Algorithms, 1990.

FK2000 DIRK FOCKEN and PURU KULKARNI, *Scribe notes, Lecture 18, Advanced Algorithms Fall 2000*