NP – Completeness

Satisfiable boolean formula

Stephen Cook and Levin discovered certain problems in NP whose individual complexity is related to that of the entire class. If a polynomial time algorithm exists for any of these problems, all problems in NP would be polynomial time solvable. These problems are called NP–complete.

One of the first NP–complete problems that were presented was the satisfiability problem. We present True by 1 and False by 0. The boolean operations AND, OR, and NOT, by the symbols $\wedge$, $\vee$, $\neg$. Notice that $\overline{x}$ means $\neg x$.

A boolean formula is an expression involving boolean variables and operations. Example:

$$\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$$

A boolean formula is satisfiable if some assignments of 0s and 1s to the variables makes the formula evaluate to 1. The previous example is satisfiable for x=0, y=1, and z=0.

The satisfiable problem is to test whether a boolean formula is satisfiable.

$$SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula}\}.$$

Polinomial time reducibility

When problem A reduces to problem B, a solution to B can be used to solve A. When problem A is efficiently reducible to problem B, an efficient solution to problem B can be used to solve A.

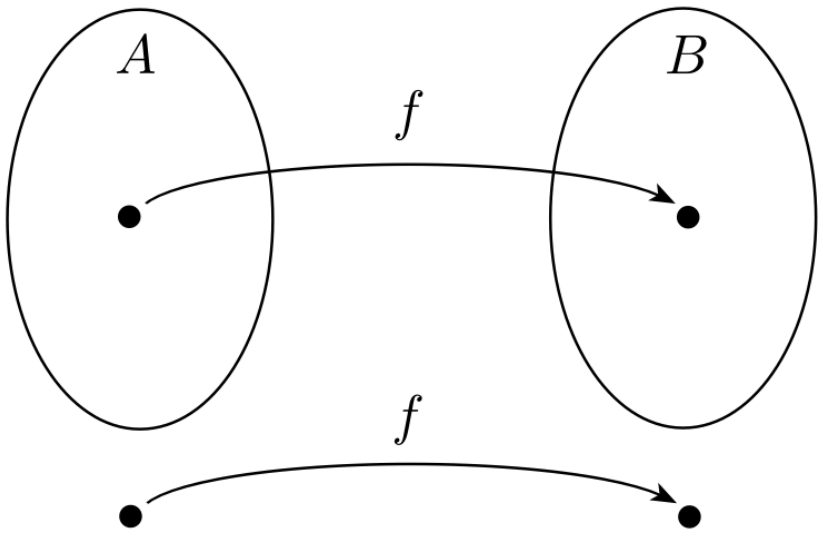
Definition:

A function $f\colon \Sigma^* \longrightarrow \Sigma^*$ is a ***polynomial time computable function*** if some polynomial time Turing machine M exists that halts with just $f(w)$ on its tape, when started on any input w.

Language A is ***polynomial time mapping reducible***,[1] or simply ***polynomial time reducible***, to language B, written $A \leq_P B$, if a polynomial time computable function $f\colon \Sigma^* \longrightarrow \Sigma^*$ exists, where for every w,

$$w \in A \Longleftrightarrow f(w) \in B.$$

The function f is called the ***polynomial time reduction*** of A to B.



Polynomial time function f reducing A to B

A polynomial time reduction of A to B provides a way to convert membership testing in A to membership testing in B. To test whether w∊A, we use the reduction function f to map w to f(w) and test whether f(w)∊B.

If one language is polynomial time reducible to a language already known to have a polynomial time solution, we obtain a polynomial time solution to the original language, as in the following theorem:

If $A \leq_P B$ and $B \in$ P, then $A \in$ P.

PROOF Let M be the polynomial time algorithm deciding B and f be the polynomial time reduction from A to B. We describe a polynomial time algorithm N deciding A as follows.

```
N = "On input w:
        1. Compute f(w).
        2. Run M on input f(w) and output whatever M outputs."
```

We have $w \in A$ whenever $f(w) \in B$ because f is a reduction from A to B. Thus, M accepts $f(w)$ whenever $w \in A$. Moreover, N runs in polynomial time because each of its two stages runs in polynomial time. Note that stage 2 runs in polynomial time because the composition of two polynomials is a polynomial.

(3)CNF – formula

3SAT is a special case of the satisfiability problem where all formulas are in a special form.

A ***literal*** is a boolean variable, as in x or $\overline{x}$. A ***clause*** is several literals connected with ∨s. A boolean formula is in ***conjunctive normal form***, called a ***cnf-formula***, if it comprises several clauses connected with ∧s.

It is ***3cnf-formula*** if all the clauses have three literals, as in:

$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4) \wedge (x_4 \vee x_5 \vee x_6).$$

Let *3SAT* $= \{\langle\phi\rangle \mid \phi$ is a satisfiable 3cnf-formula$\}$

Now we prove that 3SAT is polynomial time reducible to *CLIQUE*.

The polynomial time reduction function that we demonstrate from 3SAT to CLIQUE will convert formulas to graphs.

PROOF Let ϕ be a formula with k clauses such as

$$\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \cdots \wedge (a_k \vee b_k \vee c_k).$$

The reduction generates <G, k>, where G is constructed as follows:

G has k groups of 3 nodes each, corresponding to each clause and each literal. There are two rules for connecting nodes. All nodes are connected except:

- Nodes from the same clause (triple)
- Nodes with contradictory labels (x and $\overline{x}$).

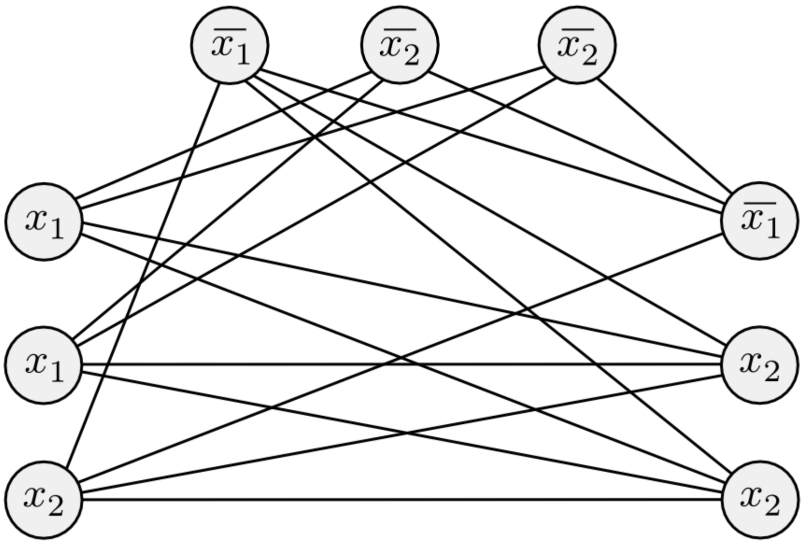


FIGURE 7.33

The graph that the reduction produces from

$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

We show that Ø is satisfiable **iff** G has k-clique.

Suppose Ø is satisfiable. We have at least 1 true literal in each clause. Select one random true literal from each clause. The nodes selected can't be from the name clause since we selected one from each and they can't have contradictory labels, since we chose only true literals. Thus, it is a k-clique ⇒ G has a k-clique.

Suppose G has a k-clique. Assign the value true to each of the literals in the clique. They can't be from the same triple and can't be contradictory. Thus, we created a formula Ø which has at least one true variable in each of its clauses ⇒ Ø is satisfiable.

NP – complete

A language B is NP–complete if it satisfies two conditions:

- B is in NP
- Every A in NP is polynomial time reducible to B

If B is NP–complete and B∊P, then P=NP.

If B is NP–complete and $B \leq_P C$ for C in NP, then C is NP–complete.

B is NP–complete |

$A \leq_P B$ | ⇒ $A \leq_P C$ ⇒ C is NP–complete

$B \leq_P C$ |

Cook–Levin Theorem

Theorem – SAT is NP–complete

…

RIP 🧯🔥

3SAT is NP – complete

RIP 🧯🔥