NP COMPLETE PROBLEMS

DR. GODFREY C. MUGANDA

1. Decision Problems

A *decision problem* is a computational problem in which an algorithm is being asked to answer a YES-NO question.

In a decision problem, an algorithm is being asked to *decide* whether a certain statement about the inputs is true (YES) or false (NO).

Many problems can be studied through a related decision problem.

Consider the computational problem

Integer Sum
Input: Two integers A and B.
Output: An integer S such that $S = A + B$.

The related decision problem is

Integer Sum
Input: Three integers A, B and S.
Question: Is $S = A + B$?

One reason we are interested in decision problems, is that we can show that a computational problem is hard (requires a lot of resources, or cannot be solved in polynomial time) by showing that the associated decision problem is hard.

2. YES instances and NO instances

Given a decision problem X, some instances of X will be YES-instances, and other instances of X will be NO-instances.

In the Integer Sum problem,

(1) an instance with input $A = 5$, $B = 7$ and $S = 12$ is a YES instance,
(2) an instance with input $A = 5$, $B = 7$ and $S = 10$ is a NO instance.

Consider the decision problem for array sorting:

Array Sorting
Input: An integer array $a[1, \ldots, n]$.
Question: Is the array $a[1, \ldots, n]$ sorted?

For this problem:

(1) an instance with input $a[\,] = [5, 12, 17]$ is YES instance
(2) an instance with input $a[\,] = [5, 5, 3, 1]$ is a YES instance,
(3) but an instance with input $a[\,] = [5, 6, 3]$ is a NO instance.

3. Famous Decision Problems

HAMILTONIAN PATH
INPUT: A directed graph $G = (V, E)$
QUESTION: Does G have a Hamiltonian path?

HAMILTONIAN CYCLE
INPUT: A directed graph $G = (V, E)$
QUESTION: Does G have a Hamiltonian cycle?

CLIQUE
INPUT: A graph $G = (V, E)$ and an integer k
QUESTION: Does G have a clique of size k?

INDEPENDENT SET
INPUT: A graph $G = (V, E)$ and an integer k
QUESTION: Does G have an independent set of size k?

VERTEX COVER
INPUT: A graph $G = (V, E)$ and an integer k
QUESTION: Does G have a vertex cover of size k?

4. P and NP

There are two important classes of decision problems, called P and NP.

P stands for Polynomial Time.

A decision problem is in P if the associated question can be answered in polynomial time.

NP stands for Non-deterministic Polynomial Time.

A decision problem is in NP if for every YES-instance of the problem, you can guess a YES-certificate and verify the certificate in polynomial time.

A YES certificate is a data structure that can serve as proof of the "YES-ness" of an instance.

Best illustration of this is Hamiltonian Path. If a graph has a Hamiltonian path, the Hamiltonian path is the YES-certificate, and a skeptical person can verify in polynomial time that the list of vertices that is claimed to be a Hamiltonian path is indeed a Hamiltonian path.

If it is a NO-instance, there is no known way to verify that there is no Hamiltonian path in polynomial time.

So the meaning of NP is this: If it is a YES instance, you can guess a YES certificate and verify it in polynomial time.

5. Polynomial Reductions Among Decision Problems

Let X and Y be decision problems.

X is *polynomially reducible* to Y if there is a polynomial-time algorithm that transforms instances of X into instances of Y in such a way that YES-instances of X are transformed into YES-instances of Y and NO-instances of X are transformed into NO-instances of Y.

This means that if you have a polynomial time algorithm for solving Y, you can use it to solve X in polynomial time.

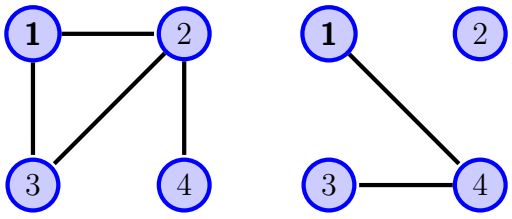
6. Examples of Polynomial Reductions

It turns out that CLIQUE, INDEPENDENT SET and VERTEX COVER are all polynomially reducible to each other.

7. 2-Way Reduction between Clique and Independent Set

Let $G = (V, E)$ be an undirected graph. The *complement* of G is a graph $\overline{G}$ with the same vertex set as G, and whose edge set consists of exactly those edges that are not in G.

Here is an example of a graph G on the left, and its complement $\overline{G}$ on the right.



Notice that a clique in G is an independent set in $\overline{G}$ and vice versa:

$\{1, 2, 3\}$ is a clique in G
$\{1, 2, 3\}$ is an independent set in $\overline{G}$

8. 2-Way reduction between Independent Set and Vertex Cover

Let $G = (V, E)$ be an undirected graph, and let $I \subseteq V$ be a set of vertices in G. Then I is an independent set in G if and only if its complement $V \setminus I$ is a vertex cover.

Proof: let I be an independent set. Then no edge has both endpoints in I. Thus every edge must have at least one of its endpoints in $V \setminus I$. Therefore $V \setminus I$ is a vertex cover.

Conversely, if C is a vertex cover, every edge has at least one of its endpoints in C, so no edge has both endpoints in $V \setminus C$, which must then be and independent set in G.

This shows that all three problems are really the same problem. We now introduce one more decision problem.

9. Clauses and Literals

Given a set of boolean variables $P, Q, \ldots$ etc, We define a *literal* to be an expression that is either a boolean variable or the negation of a boolean variable.

Examples of literals:

$$P, \quad \neg P, \quad \neg Q.$$

A *disjunction* of literals is called a *clause*. Examples of clauses:

$$P, \quad P \vee Q, \quad \neg P \vee Q, \quad \neg Q \vee \neg P$$

Note that $P \implies Q$ is equivalent to $\neg P \vee Q$ so we will think of it as a clause.

A set of clauses is satisfiable if there exists an assignment of truth values T and F to each boolean variable that makes all the clauses true.

For example, the set of clauses

$$P \vee Q, \quad \neg Q$$

is satisfiable because $P = T$ and $Q = F$ is a satisfying assignment.

Here is a non-satisfiable set of clauses.

$$\neg P, \quad Q, \quad Q \implies P$$

10. SATISFIABILITY

SATISFIABILITY, usually abbreviated SAT, is the following decision problem:

SATISFIABILITY
INPUT: A set of clauses.
QUESTION: Is there a truth assignment that satisfies all the clauses?

What is the significance of SAT? It turns out that *every* problem in NP is polynomially reducible to SAT.

11. REDUCTION OF HAMILTONIAN PATH TO SAT

Let $G = (V, E)$ be an instance of the graph being tested for the existence of a Hamiltonian path.

We are going to build an instance of SAT that is satisfiable if and only if G has a Hamiltonian path. We assume that $V = \{1, 2, \ldots, n\}$.

For each pair of vertices u and v, use a propositional variable $E[u, v]$ to represent the presence of the edge $u \to v$. Create a literal

$$E[u, v]$$

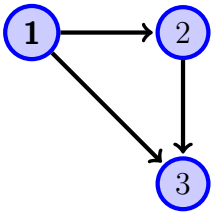
for each edge $u \to v$, in G, and create a literal

$$\neg E[u, v]$$

when there is no such edge.

You can think of the $E[\cdot, \cdot]$ literals as defining the adjacency matrix of the graph, and hence, its pattern of edge connectivity.

As an example, if $n = 3$ and we have the simple digraph shown here, the $E[\cdot, \cdot]$ literals will be as shown.



$\neg E[1,1]$	$E[1,2]$	$E[1,3]$
$\neg E[2,1]$	$\neg E[2,2]$	$E[2,3]$
$\neg E[3,1]$	$\neg E[3,2]$	$\neg E[3,3]$

Now we need to create a bunch of clauses that all together will be true if and only if there is a Hamiltonian path in G.

To have a Hamiltonian path in a graph of n vertices, we must place exactly one vertex in each of the positions $1, 2, \ldots, n$ of a list in such a way that if a vertex u is in position i and a vertex v is in position $i+1$, then there must be an edge $u \to v$.

For each vertex v and each list position i, we introduce a propositional variable

$$P[v, i]$$

whose interpretation is that vertex v occupies position i on the list.

We now use these variables to create literals that state that there is at most one vertex in each list position. For each vertex $v = 1, \ldots, n$ and position $i = 1, \ldots n$, we will have a clause

$$P[v, i] \implies \neg P[u, i] \qquad (\text{for all } u \neq v)$$

For our example graph with 3 vertices, we will have the following set of literals to assert that vertex 1 cannot share its position with any other vertex. For example, the first column says that if vertex 1 is in position 1, then vertices 2 and 3 cannot also be in position 1.

$$\begin{array}{lll} P[1,1] \implies \neg P[2,1] & P[1,2] \implies \neg P[2,2] & P[1,3] \implies \neg P[2,3] \\ P[1,1] \implies \neg P[3,1] & P[1,2] \implies \neg P[3,2] & P[1,3] \implies \neg P[3,3] \end{array}$$

Naturally, we need to add two similar sets of clauses, to assert that vertex 2 cannot share its position, and also that vertex 3 cannot share its position, with any other vertex.

Now we need sets of clauses to assert that there is a vertex at each position i. For each position i, we need a clause

$$P[1,i] \vee P[2,i] \vee \cdots \vee P[n,i].$$

For our specific example of 3 vertices, we will have a set of clauses

$$\begin{array}{c} P[1,1] \vee P[2,1] \vee P[3,1] \\ P[1,2] \vee P[2,2] \vee P[3,2] \\ P[1,3] \vee P[2,3] \vee P[3,3] \end{array}$$

Finally, for each position $i = 1, \ldots, n-1$, add a clause

$$P[v,i] \wedge P[u,i+1] \implies E[v,u]$$

which says that if vertices v and u are in successive positions in the list order, there must be an edge from v to u.

Call these the *path clauses*. For our graph of three vertices, the path clauses are

$$\begin{array}{llll} P[1,1] \wedge P[2,2] \implies E[1,2] & & P[1,1] \wedge P[3,2] \implies E[1,3] \\ P[1,2] \wedge P[2,3] \implies E[1,2] & & P[1,2] \wedge P[3,3] \implies E[1,3] \\ P[2,1] \wedge P[1,2] \implies E[2,1] & & P[2,1] \wedge P[3,2] \implies E[2,3] \\ P[2,2] \wedge P[1,3] \implies E[2,1] & & P[2,2] \wedge P[3,3] \implies E[2,3] \\ P[3,1] \wedge P[1,2] \implies E[3,1] & & P[3,1] \wedge P[2,2] \implies E[3,2] \\ P[3,2] \wedge P[1,3] \implies E[3,1] & & P[3,2] \wedge P[2,3] \implies E[3,2] \end{array}$$

It so happens our graph has a Hamiltonian path, which can be represented as $P[1,1] = T$, $P[2,2] = T$ and $P[3,3] = T$ and setting all other $P[v,i] = F$.

Clearly, all our clauses are satisfied. Conversely, if all our clauses are satisfied, it is clear that G will have a Hamiltonian path.

All these clauses can be built in polynomial time. Thus HAMILTONIAN PATH is polynomially reducible to SAT.

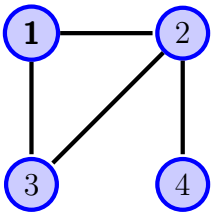
12. Reduction of Clique to SAT

Hamiltonian path is a sequence. Let us now take a look at how to use SAT to describe a set, as in CLIQUE.

CLIQUE
INPUT: And undirected graph $G = (V, E)$ where $V = \{1, \ldots, n\}$, and an integer k where $1 \leq k \leq n$.
QUESTION: Does G have a clique of size k?

The process is similar to HAMILTONIAN PATH: we use a set of clauses to describe the adjacency structure of the graph, and then build a set of clauses that will be satisfiable if and only if the instance of the graph has a clique of size k.

In our example, $n = 4$ and $k = 3$, so we are asking if the graph has a clique of size 3.



$\neg E[1,1]$	$E[1,2]$	$E[1,3]$	$\neg E[1,4]$
$E[2,1]$	$\neg E[2,2]$	$E[2,3]$	$E[2,4]$
$E[3,1]$	$E[3,2]$	$\neg E[3,3]$	$\neg E[3,4]$
$\neg E[4,1]$	$E[4,2]$	$\neg E[4,3]$	$\neg E[4,4]$

Now we need a set of clauses that assert that there is a set of k vertices, and an additional set of clauses that assert that those k vertices form a clique.

Think of a set of k vertices as a list of vertices with a vertex at positions 1, 2, ... k. For each vertex v and position $i = 1, 2, \ldots, k$, define

$$S[v, i]$$

to mean that we select vertex v as the $i$the member of the set of k vertices.

After that, we add the following clauses that assert that there is at least one vertex selected for each position.

$$\begin{array}{ll}
\text{There is a vertex at position 1 :} & S[1,1] \vee S[2,1] \vee \cdots \vee S[n,1] \\
\text{There is a vertex at position 2 :} & S[1,2] \vee S[2,2] \vee \cdots \vee S[n,2] \\
\vdots & \vdots \\
\text{There is a vertex at position k:} & S[1,k] \vee S[2,k] \vee \cdots \vee S[n,k]
\end{array}$$

Specializing these clauses to our example, we have

$$\begin{array}{c}
S[1,1] \vee S[2,1] \vee S[3,1] \vee S[4,1] \\
S[1,2] \vee S[2,2] \vee S[3,2] \vee S[4,2] \\
S[1,3] \vee S[2,3] \vee S[3,3] \vee S[4,3]
\end{array}$$

Then we need clauses stating that there is at most one vertex at each position. For each $i = 1, \ldots, k$ and $v \neq u$, we have a clause

$$S[v,i] \implies \neg S[u,i]$$

For our example, some of these clauses will be

$$\begin{array}{llll}
S[1,1] \implies \neg S[2,1] & S[2,1] \implies \neg S[1,1] \\
S[1,1] \implies \neg S[3,1] & S[2,1] \implies \neg S[3,1] \\
S[1,1] \implies \neg S[4,1] & S[2,1] \implies \neg S[4,1]
\end{array}$$

First column says that if vertex 1 is selected for position 1, then vertices 2, 3 and 4 cannot be selected for that position.

Likewise column 2 says that if vertex 2 is selected for position 1, then vertices 1, 3 and 4 cannot be selected for that position.

We add similar clauses for vertices 3 and 4 and positions 2 and 3.

Finally we want a set of clauses that says that the k selected vertices form a click. Add a clause

$$S[v,i] \wedge S[u,j] \implies E[u,v]$$

for all $i \neq j$.

Again this shows that CLIQUE is polynomially reducible to SAT.

13. NP-Complete problems

A problem X in NP is *NP-complete* if every other problem in NP is polynomially reducible to X.

In 1974, Stephen Cook at the University of Toronto proved that SAT is NP-complete.

This means that every problem in NP is can be transformed into a special case of SAT. therefore, if there is a polynomial time algorithm for SAT, there is a polynomial algorithm for every problem in NP.

There are many problems in NP, like Hamiltonian Path, Hamiltonian Cycle , Clique, Vertex Cover, Independent Set for which no polynomial time algorithms are known.

Turns out that all these problems are also NP-complete, which means all of them are polynomially equivalent. If one of them has a polynomial time algorithm, they all do, and $NP = P$.

The question is $NP = P$? is the outstanding open problem in Computer Science today.

There is a million dollar prize offered for its solution.

Most computer scientists believe that $P \neq NP$.