NP-completeness

Bruno MARTIN,
University of Nice - Sophia Antipolis
mailto:Bruno.Martin@unice.fr
http://deptinfo.unice.fr/~bmartin/mathmods.html

Definition of P

P: decision problems for which there is a polytime algorithm.

Problem	Description	Algorithm	Yes	No
Multiple	is x a multiple of y	division	51, 17	51, 16
Rel. prime	$\gcd(x, y) = 1$?	Euclid	34, 39	34, 51
Primes	is x prime?	AKS'02	53	51
Isolve	$\exists ?x$ that satisfies $Ax = b$?	Gauss Edmonds	$\begin{bmatrix} 0 & 1 & 1 \\ 2 & 4 & -2 \\ 0 & 3 & 15 \end{bmatrix} \begin{bmatrix} 4 \\ 2 \\ 36 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$

Decision problem

- Π is a set of strings (a language)
- Instance: string s over a finite alphabet Σ
- Algorithm A decides problem Π: $A(s) =$ yes iff $s \in \Pi$

A runs in polynomial time if for every string s, $A(s)$ terminates in at most $p(\sharp s)$ steps, where p is some polynomial.

Example

PRIMES: $\Pi = \{2, 3, 5, 7, 11, 13, 17, 23, 29, 31, 37, \ldots\}$
Algorithm [Agrawal, Kayal and Saxena, 2002] $p(\sharp s) = \sharp s^8$

NP : Non-deterministic polynomial time

Through the intuition of a certification algorithm;

- views things from a "boss" viewpoint
- doesn't determine whether $s \in \Pi$ on its own; rather it checks a proposed (short enough) proof/certificate t that $s \in \Pi$

Definition

$C(s, t)$ is a **certifier** for Π if $\forall s \in \Pi, \exists t$ st $C(s, t) =$ yes ($t=$ certificate or witness)

NP : decision problems for which there is a polytime certifier

Certifiers and certificates: composites

COMPOSITES: given $s \in \mathbb{N}$, is s composite?
Certificate: a nontrivial factor t of s. Note that such a certificate exists iff s composite. Moreover, $\sharp t \leq \sharp s$

Certifier

```
def C(s,t)
 if t<=1 or t>=s
  return false
 elsif s is a multiple of t
  return true
 else
  return false
 end
end
```

Instance $s = 437\,669$
Certificate $t = 541$ or 809

Thus, COMPOSITES is in NP

A brief history of complexity

- **Key problem**: TSP; Karp tried to solve TSP in the 60's.
- In the 60's, complexity theory was introduced by Rabin, McNaughton, Yamada and Hartmanis, Stearns introduced the word complexity in 1965 with a model of computation and the first results on the structure of complexity classes.
- In the 60's, Edmonds introduced the notion of good algorithm as a polytime algorithm on the size of the problem encoding.
- P and NP were introduced in 1971 by Cook who proved that SAT is NP -complete and that all NP -complete problems reduce to SAT. TSP is among those problems and there's no hope for finding an efficient algorithm for solving TSP.
- Karp introduces the notion of reduction to prove that 21 problems are NP -complete
- Since then, a million-$ conjecture is to decide wether

$$\mathrm{P} = \mathrm{NP}$$

P vs NP

$\mathrm{P} \subseteq \mathrm{NP}$: polytime algorithm particular case of a certifier ($t = \varepsilon$).
What about the converse?

Theorem

If $\Pi \in$ NP , $s \in \Pi$ s of size n can be decided by an algorithm in time $O(2^{p(n)})$.

Proof: For every string $s \in \Sigma^n$ accepted by a certifier, there is a polynomial p and a certificate $t \in \Sigma^{p(n)}$ s.t. time$(C(s,t)) \leq p(n)$. We can generate all the t possible strings and test whether $C(s,t)$ is true within $p(n)$ steps. The overall running time of this algorithm is $p(n)\sharp\Sigma^{p(n)} = O(2^{q(n)})$ for a polynomial q

Polynomial transformation

Definition

Problem X polytime reduces (Cook) to problem Y if arbitrary instances of X can be solved using:

- polytime number of standard computation step, plus
- polytime number of calls to oracle that solves Y

Definition

Problem X polytime transforms (Karp) to problem Y ($X \propto Y$) if given any input x to X, we can construct an input $y = f(x)$ to Y st x is a yes instance of X iff y is a yes instance of Y with $\sharp y$ polynomial in $\sharp x$ and f polytime computable.

Some properties

Lemma

If $X \propto Y$ then,

1. *$Y \in \mathrm{P}$ implies $X \in \mathrm{P}$*
2. *$X \notin \mathrm{P}$ implies $Y \notin \mathrm{P}$*

1. If $A \in \mathrm{P}$ decides Y, since $X \propto Y$, one can design B a polytime algorithm for deciding X: $y \in Y$ with $A(y)$ =yes, $B(x) = A(f(x))$
2. assume $A \in \mathrm{P}$ decides Y. Since $X \propto Y$, one can design $B \in \mathrm{P}$ for deciding X: let $x \in X$ and $y = f(x) \in Y$. $B(x) = A(f(x))$ and since $A \in \mathrm{P}$ and $f \in \mathrm{P}$, $X \in \mathrm{P}$, a contradiction.

Lemma (Transitivity)

If $X \propto Y$ and $Y \propto Z$, then $X \propto Z$

Howto: Establishing NP -completeness of Π

We should prove that any problem in NP transforms to Π...

But once we've established a first "natural NP -complete" problem, other fall like dominoes since:

Lemma

Let $X \in$ NP , $Y \in$ NP . If X is NP -complete and $X \propto Y$, then Y is NP -complete.

Recipe to establish NP -completeness of Π:

1. show that $\Pi \in$ NP
2. choose a NP -complete problem X
3. prove $X \propto \Pi$

NP -completeness

Definition

Y is NP -complete if $Y \in$ NP with the property that for every problem $X \in$ NP , $X \propto Y$.

Theorem

Suppose Y NP-complete. Then Y is polytime decidable iff $\mathrm{P} = \mathrm{NP}$

- $\Leftarrow$ If P = NP , then Y polytime solvable since $Y \in$ NP
- $\Rightarrow$ Suppose Y can be solved in polytime.
  - Let X be any problem in NP . Since $X \propto Y$, we can solve X in polytime. This implies NP $\subseteq P$
  - We already know $\mathrm{P} \subseteq \mathrm{NP}$ thus $\mathrm{P} = \mathrm{NP}$

Are there any "natural" NP -complete problems?

The first NP -complete problem

Cook proved that the satisfiability problem is NP -complete:

INSTANCE : U set of variables; C, collection of clauses over U
QUESTION : Does there exist a valuation which satisfies C?

Theorem (Cook)

SAT is NP -complete

But another kind of reduction and the precise notion of a computation model are required to prove this.

Satisfiability problem

$U = \{u_1, u_2, \ldots, u_n\}$ a set of variables
$t : U \longrightarrow \{0, 1\}$ a *truth assignment* of the variables of U
$t(u) = 1$ iff u is true and $t(u) = 0$ iff u is false.
For $u \in U$, u and $\overline{u}$ are literals.
u is true by t iff $t(u) = 1$ and $\overline{u}$ is true by t iff $t(u) = 0$.
A *clause* C is a set of literals which is the disjunction of the literals.

Example

$\{u_1, \overline{u_3}, u_8\} \Leftrightarrow u_1 \vee \neg u_3 \vee u_8$ true for $t(u_1) = 1$ or $t(u_3) = 0$ or $t(u_8) = 1$.

A set of clauses is satisfiable iff there exists a truth assignment for U satisfying simultaneously all the clauses of C. Equiv., if there is a truth assignment which satisfies the conjunction of the clauses.

3-SAT

INSTANCE : A collection $C = \{c_1, c_2, \ldots, c_m\}$ of clauses over a finite set of variables U such that for all i, $|c_i| = 3$
QUESTION : Does there exist a truth assignment of U which satisfies simultaneously all the clauses of C?

Theorem

3-SAT is NP *-complete.*

3-SAT$\in$ NP : Given a truth assignment of U, the satisfiability of the formula can be checked by a polytime algorithm.

Example

Let $U = \{u_1, u_2\}$ and $C = (\{u_1, \overline{u_2}\}, \{\overline{u_1}, u_2\})$ or equivalently, $(u_1 \vee \neg u_2) \wedge (u_2 \vee \neg u_1)$.
This is a yes-instance for the next truth assignment:

u_1	u_2	$(u_1 \vee \neg u_2) \wedge (u_2 \vee \neg u_1)$
0	0	1
1	1	1
0	1	0
1	0	0

SAT$\propto$3-SAT

We consider an instance of SAT
$U = \{u_1, \ldots, u_n\}$ set of variables and
$C = \{c_1, \ldots, c_m\}$ set of clauses
We build a collection C' of clauses of 3 literals over a set U' of variables such that C' is satisfiable iff C is satisfiable.
We define the variables of 3-SAT:
Each clause $c_j \in C$ will be represented by an equivalent collection of clauses c'_j of three literals which will use the original variables from U which occur in c_j and auxiliary variables from U'_j which will be used only by clauses c'_j. Thus,

$$U' = U \cup \left(\cup_{j=1}^{m} U'_j\right) \text{ and } C' = \cup_{j=1}^{m} c'_j$$

Clauses of 3-SAT

We build c'_j and U'_j from $c_j = \{z_1, z_2, \ldots, z_k\}$, where the z_i's are literals derived from the variables in U. To build U'_j and c'_j, there are several cases depending on the value of k (number of literals):

1. $k = 1$: c_j has a single literal; $U'_j = \{y_j^1, y_j^2\}$ and
$$c'_j = \{\{z_1, y_j^1, y_j^2\}, \{z_1, y_j^1, \overline{y_j^2}\}, \{z_1, \overline{y_j^1}, y_j^2\}, \{z_1, \overline{y_j^1}, \overline{y_j^2}\}\}$$
or, more easily, we can have anything!
2. $k = 2$: In this case, $U'_j = \{y_j^1\}$ and
$$c'_j = \{\{z_1, z_2, y_j^1\}, \{z_1, z_2, \overline{y_j^1}\}\}$$
3. $k = 3$: This is the simplest case since c_j already is a clause of 3 literals; thus $U'_j = \varnothing$ and $c'_j = \{\{c_j\}\}$
4. $k > 3$: more difficult: $U'_j = \{y_j^i : 1 \le i \le k-3\}$ and
$$c'_j = \{\{z_1, z_2, y_j^1\}\} \cup \{\{\overline{y_j^i}, z_{i+2}, y_j^{i+1}\} : 1 \le i \le k-4\} \cup \{\overline{y_j^{k-3}}, z_{k-1}, z_k\}$$

$\models C \Leftarrow \models C'$

Conversely, if t' satisfies C', we check that the restriction of t' to the variables of U also satisfies C. We have proved $\models C \Leftrightarrow \models C'$.

Rest to check that the transformation is polynomial.

It suffices to count the number of 3-clauses in C' which is upper-bounded by a polynomial in mn. Thus the size of the instances of 3-SAT is upper-bounded by a polynomial function in the size of SAT instances. Since all the details of the construction are immediate, we have a polynomial transformation from SAT to 3-SAT.

$\models C \Rightarrow \models C'$

$t \models C$. t can be extended in $t' \models C'$: since the variables in $U' \setminus U$ are partitioned into U'_j and the variables in every U'_j only occur in the clauses of c'_j, we just show how to extend t to the U'_j 1 by 1.

- U'_j comes from case 1. or 2. t is extended in t' arbitrarily, like $t'(y) = 1, \quad \forall y \in U'_j$.
- U'_j comes from case 3. $t = t'$
- U'_j comes from case 4. Let $c_j = \{z_1, z_2, \ldots, z_k\}$ with $k > 3$. Since $t \models c_j$, there exists ℓ such that $t(z_\ell) = 1$.
  - $\ell = 1, 2$: $t'(y_j^i) = 0$, $1 \le i \le k-3$
  - $\ell = k-1, k$: $t'(y_j^i) = 1$, $1 \le i \le k-3$
  - otherwise : $t'(y_j^i) = 1$, $1 \le i \le \ell - 2$ & $t'(y_j^i) = 0$, $\ell - 1 \le i \le k-3$

We easily check that for all these choices, all the clauses of c'_j are satisfied and thus that $t' \models c'_j$.

And for 2-SAT?

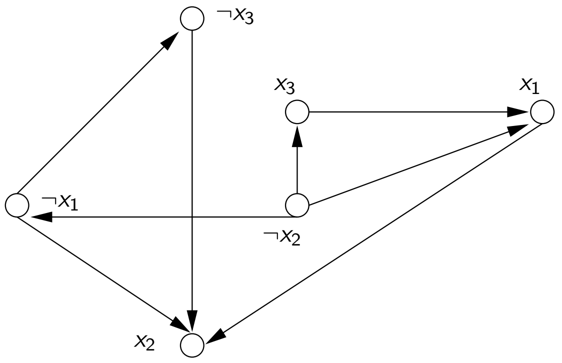
INSTANCE : ϕ a boolean formula in CNF with clauses of degree exactly 2.
QUESTION : is ϕ satisfiable?

Theorem

2-SAT $\in$ P

We build a graph whose vertices are the variables and the negation of the variables and such that for every clause $l_i \vee l_j$ we have an implication $\neg l_i \to l_j$ and $\neg l_j \to l_i$. We then compute the transitive closure of the graph by a polytime algorithm.

Graph associated with $\phi = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee x_2) \wedge (x_2 \vee x_3)$



Some NP -complete problems

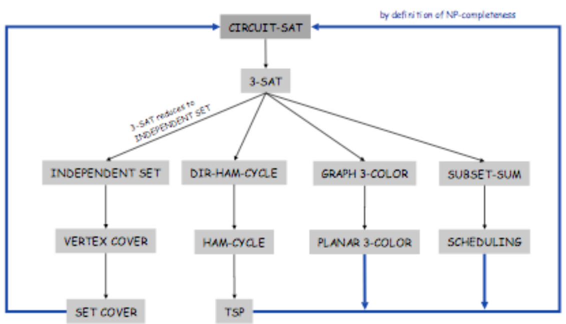
6 basic kinds of NP-complete problems and paradigmatic samples:

- Packing problems: SET-PACKING, INDEPENDANT-SET
- Covering problems: SET-COVER, VERTEX-COVER
- Constraint satisfaction problems: SAT, 3-SAT
- Sequencing problems: HAM-CYCLE, TSP
- Partitioning problems: 3D-MATCHING, 3-COLOR
- Numerical problems: SSP, KNAPSACK

Practice: most NP problems are either in P or NP -complete.
Notable exceptions: Factoring, graph isomorphism, Nash equilibrium

NP-completeness

All problems below are NP -complete and reduce to one another.



Asymmetry of NP

We only need to have short proofs of yes-instances

Example

SAT vs TAUTOLOGY

- Can prove a CNF formula is satisfiable by giving a truth assignment
- How can we prove that a formula is not satisfiable?

SAT is NP -complete and proved polynomially equivalent with TAUTOLOGY but how can we classify TAUTOLOGY which is not even known to be in NP ?

NP and co-NP

NP: Decision problems for which there is a polytime certifier

Definition

Given a decision problem Π, its complement $\overline{\Pi}$ is the same problem with the yes and no answers reverse.

Example

$$\begin{aligned} \overline{X} &= \{0, 1, 4, 6, 8, 9, 10, 12, 14, 15, \ldots\} \\ X &= \{2, 3, 5, 7, 11, 13, 17, 23, 29, \ldots\} \end{aligned}$$

co-NP : complements of decision problems in NP
Ex: TAUTOLOGY, PRIMES,...

Good characterizations

- If $X \in$ NP $\cap$ co-NP then:
  - for yes instance, there is a succinct certificate
  - for no instance, there is a succinct disqualifier
- P $\subseteq$ NP $\cap$ co-NP
- Fundamental open question: does P $=$ NP $\cap$ co-NP ?
  - Mixed opinions
  - Many examples where problem found to have a non-trivial good characterization, but only years later discovered to be in P
    - Linear Programming by Khachiyan, 1979
    - Primality testing by Agrawal, Kayal and Saxena, 2002

Fact: Factoring is NP $\cap$ co-NP but not known to be in P.

NP $=$ co-NP ?

Fundamental question: Does NP $=$ co-NP ?

- do yes-instances have succinct certificate iff no-instances do?
- consensus opinion: no

Theorem

If NP $\neq co -$ NP *, then* P $\neq$ NP .

- P is closed under complement
- if P $=$ NP , then NP closed under complement
- equivalently, NP $=$co-NP
- This is the contrapositive of the theorem

Primes $\in$ NP $\cap$ co-NP

Already known: Primes$\in$co-NP. Suffices to prove that Primes$\in$NP .

Theorem (Pratt)

An odd integer s is prime iff there exists an integer $1 < t < s$ s.t for all prime divisors p of $s - 1$,

$$\begin{aligned} t^{s-1} &\equiv 1 \mod s \\ t^{(s-1)/p} &\not\equiv 1 \mod s \end{aligned}$$

Certificate and certifier

- Input $s = 437\,677$
- Certificate: $t = 17$ and a prime factorization of $s - 1 = 2^2.3.36\,473$ which also needs a recursive certificate to guarantee that 3 and 36 473 are primes
- Certifier
  - check $s - 1 = 2^2.3.36\,473$
  - check $17^{(s-1)/2} \equiv 437676 \mod s$
  - check $17^{(s-1)/3} \equiv 329415 \mod s$
  - check $17^{(s-1)/36473} \equiv 305452 \mod s$

  by using repeated squaring algorithm