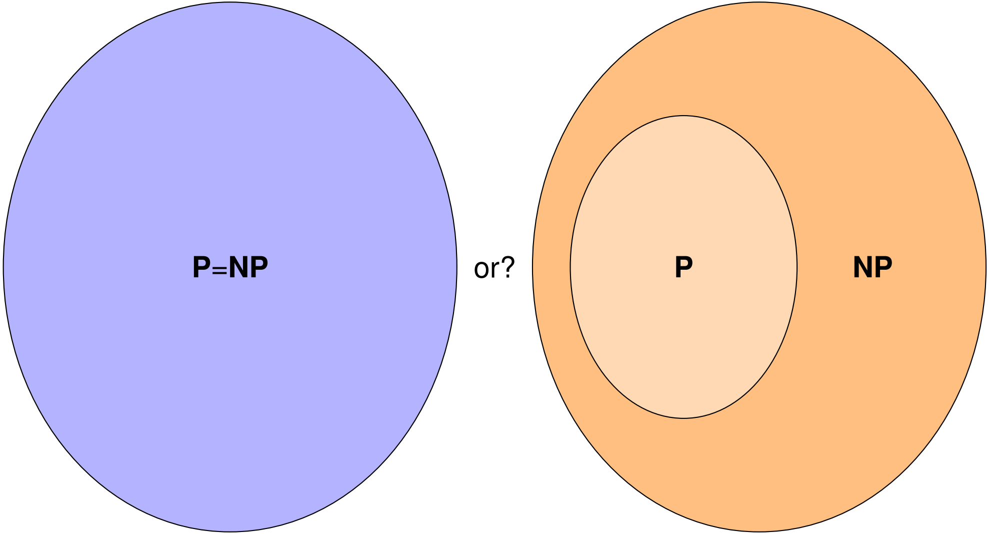
NP-Completeness

Dr Kamal Bentahar

School of Engineering, Environment and Computing
Coventry University

28/11/2016

Last time. . .



The satisfiability problem

Recall:

- Boolean variables (*True*/*False*)
- Boolean operations ($\wedge, \vee, \neg$)
- Boolean formula, e.g.

$$
\begin{aligned}
& x_1 \\
& x_1 \wedge x_2 \\
& x_1 \vee \neg x_2 \\
& \neg x_1 \wedge (x_1 \vee x_2) \\
& (x_2 \vee \neg x_3) \wedge (x_1 \vee x_2)
\end{aligned}
$$

- "**Satisfiable**" if formula can be *True* for some variables assignment.

The satisfiability problem (SAT)

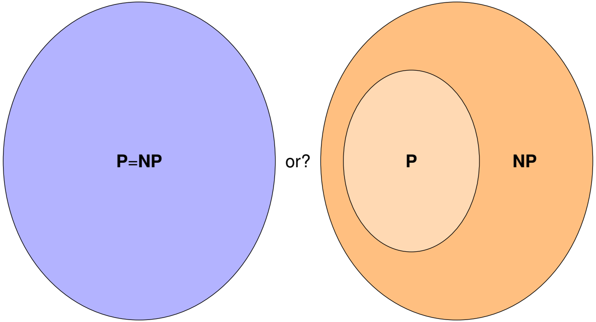
$$SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$$

Link between *SAT* and the "**P**vs **NP**" question

Theorem (Cook 1971)

$$\mathit{SAT} \in \mathbf{P} \qquad \Longleftrightarrow \qquad \mathbf{P} = \mathbf{NP}$$

i.e. if we can efficiently solve *SAT* then we can also efficiently solve any **NP** problem.



History of *SAT*

- **Stephen Cook (1971):** any problem in **NP** is transformable to *SAT* in polynomial time.
  Efficient solution to *SAT* $\Longrightarrow$ Efficient solution to every problem in **NP**.
- **Richard Karp (1972):** listed 21 problems all transformable into each other in polynomial time.
- **Garey and Johnson (1979):** book *"Computers and Intractability: A Guide to the theory of NP-Completeness"* lists 320 problems all transformable into each other in polynomial time.
- These "**NP-complete**" problems are the "hardest in **NP**."
- If an **NP-complete** problem is not in **P** then all of them are not in **P**. ($\Longrightarrow$ **P** $\neq$ **NP**).

Reducibility

The aim is to transform a given problem to another, such that we can solve it by using the solution to that other problem to solve the original one.

Polynomial time computable functions

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

The function f "efficiently transforms" the encodings of the two problems.

Example (From Coursework 2)

Given a set $S = \{x_1, \ldots, x_n\}$ for **PP**, we transform it into and **SSP** isntance as follows:

- Calculate $t = (x_1 + \cdots + x_n)/2$.
- The **SSP** instance is $\langle S, t \rangle$.

Reducibility

Polynomial time reducibility between problems

A language A is polynomial time **reducible** to a language B if a polynomial time computable function $f\colon \Sigma^* \to \Sigma^*$ exists such that

$$w \in A \iff f(w) \in B \qquad \text{for all } w \in \Sigma^*$$

We write

$$A \leq_P B$$

and read it "A is (polytime) reducible to B."

In particular, note that if

$$A \leq_P B \text{ and } B \in \mathbf{P} \implies A \in \mathbf{P}$$

i.e. if A can be reduced to an "easy" problem B then A is also "easy."

NP-Completeness and **NP**-Hardness

NP-Completeness

A language C is **NP-complete** if it satisfies two conditions:

1. C is in **NP**
2. every problem in **NP** is polynomial time reducible to C.

The word "complete" is used to to mean that a solution to any problem can be applied to all others in the class.

NP-Hardness

A language C is **NP-hard** if it satisfies:

1. every problem in **NP** is polynomial time reducible to C.

So, a problem is **NP-complete** if it is **NP-hard** and is in **NP**.

Example **NP-complete** problems

The Cook-Levin Theorem

SAT is **NP-complete.**

- **Constraint Satisfaction**: SAT, 3SAT
- **Covering**: Set Cover, Vertex Cover, Feedback Set, Clique Cover, Chromatic Number, Hitting Set
- **Packing**: Set Packing
- **Partitioning**: 3D-Matching, Exact Cover
- **Sequencing**: Hamilton Circuit, Sequencing
- **Numerical Problems**: Subset Sum, Max Cut

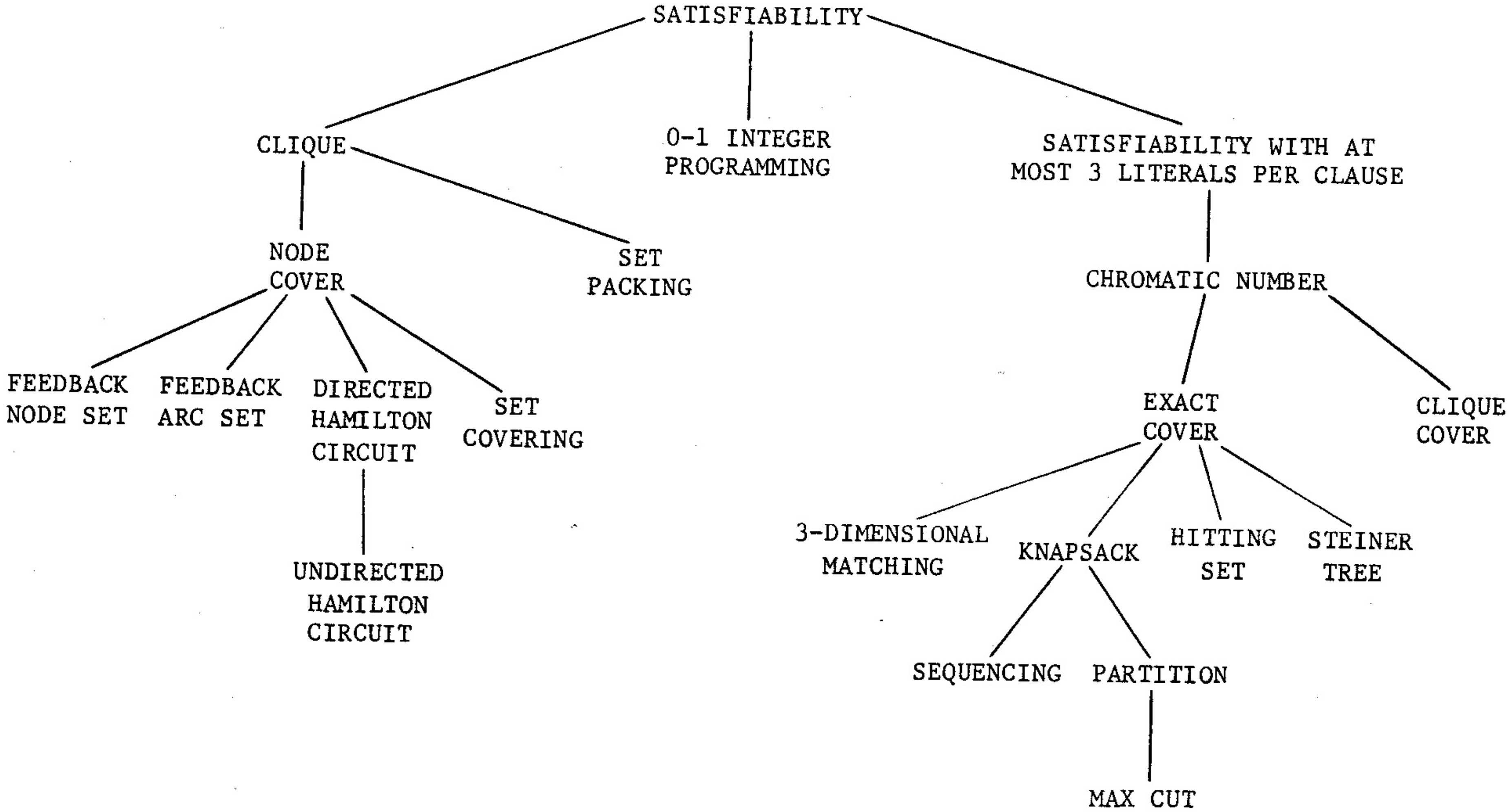


FIGURE 1 - Complete Problems

Main Theorem. All the problems on the following list are complete.

1. SATISFIABILITY
COMMENT: By duality, this problem is equivalent to determining whether a disjunctive normal form expression is a tautology.

2. 0-1 INTEGER PROGRAMMING
INPUT: integer matrix C and integer vector d
PROPERTY: There exists a 0-1 vector x such that $Cx = d$.

3. CLIQUE
INPUT: graph G, positive integer k
PROPERTY: G has a set of k mutually adjacent nodes.

4. SET PACKING
INPUT: Family of sets $\{S_j\}$, positive integer ℓ
PROPERTY: $\{S_j\}$ contains ℓ mutually disjoint sets.

5. NODE COVER
INPUT: graph G', positive integer ℓ
PROPERTY: There is a set $R \subseteq N'$ such that $|R| \leq \ell$ and every arc is incident with some node in R.

6. SET COVERING
INPUT: finite family of finite sets $\{S_j\}$, positive integer k
PROPERTY: There is a subfamily $\{T_h\} \subseteq \{S_j\}$ containing $\leq k$ sets such that $\cup T_h = \cup S_j$.

7. FEEDBACK NODE SET
INPUT: digraph H, positive integer k
PROPERTY: There is a set $R \subseteq V$ such that every (directed) cycle of H contains a node in R.

8. FEEDBACK ARC SET
INPUT: digraph H, positive integer k
PROPERTY: There is a set $S \subseteq E$ such that every (directed) cycle of H contains an arc in S.

9. DIRECTED HAMILTON CIRCUIT
INPUT: digraph H
PROPERTY: H has a directed cycle which includes each node exactly once.

10. UNDIRECTED HAMILTON CIRCUIT
INPUT: graph G
PROPERTY: G has a cycle which includes each node exactly once.

11. SATISFIABILITY WITH AT MOST 3 LITERALS PER CLAUSE
INPUT: Clauses $D_1, D_2, \ldots, D_r$, each consisting of at most 3 literals from the set $\{u_1, u_2, \ldots, u_m\} \cup \{\bar{u}_1, \bar{u}_2, \ldots, \bar{u}_m\}$
PROPERTY: The set $\{D_1, D_2, \ldots, D_r\}$ is satisfiable.

12. CHROMATIC NUMBER
INPUT: graph G, positive integer k
PROPERTY: There is a function $\phi: N \to Z_k$ such that, if u and v are adjacent, then $\phi(u) \neq \phi(v)$.

13. CLIQUE COVER
INPUT: graph G', positive integer ℓ
PROPERTY: N' is the union of ℓ or fewer cliques.

14. EXACT COVER
INPUT: family $\{S_j\}$ of subsets of a set $\{u_i,\ i = 1,2,\ldots,t\}$
PROPERTY: There is a subfamily $\{T_h\} \subseteq \{S_j\}$ such that the sets T_h are disjoint and $\cup T_h = \cup S_j = \{u_i,\ i = 1,2,\ldots,t\}$.

15. HITTING SET
INPUT: family $\{U_i\}$ of subsets of $\{s_j,\ j = 1,2,\ldots,r\}$
PROPERTY: There is a set W such that, for each i, $|W \cap U_i| = 1$.

16. STEINER TREE
INPUT: graph G, $R \subseteq N$, weighting function $w: A \to Z$, positive integer k
PROPERTY: G has a subtree of weight $\leq k$ containing the set of nodes in R.

17. 3-DIMENSIONAL MATCHING
INPUT: set $U \subseteq T \times T \times T$, where T is a finite set
PROPERTY: There is a set $W \subseteq U$ such that $|W| = |T|$ and no two elements of W agree in any coordinate.

18. KNAPSACK
INPUT: $(a_1, a_2, \ldots, a_r, b) \in Z^{n+1}$
PROPERTY: $\Sigma\, a_j x_j = b$ has a 0-1 solution.

19. JOB SEQUENCING
INPUT: "execution time vector" $(T_1, \ldots, T_p) \in Z^p$,
"deadline vector" $(D_1, \ldots, D_p) \in Z^p$
"penalty vector" $(P_1, \ldots, P_p) \in Z^p$
positive integer k
PROPERTY: There is a permutation π of $\{1,2,\ldots,p\}$ such that

$$\left(\sum_{j=1}^{p} [\text{if } T_{\pi(1)} + \cdots + T_{\pi(j)} > D_{\pi(j)} \text{ then } P_{\pi(j)} \text{ else } 0]\right) \leq k \quad .$$

20. PARTITION
INPUT: $(c_1, c_2, \ldots, c_s) \in Z^s$
PROPERTY: There is a set $I \subseteq \{1,2,\ldots,s\}$ such that $\sum_{h \in I} c_h = \sum_{h \notin I} c_h$.

21. MAX CUT
INPUT: graph G, weighting function $w: A \to Z$, positive integer W
PROPERTY: There is a set $S \subseteq N$ such that

$$\sum_{\substack{\{u,v\} \in A \\ u \in S \\ v \notin S}} w(\{u,v\}) \geq W \quad .$$

It is clear that these problems (or, more precisely, their encodings into Σ^*), are all in NP. We proceed to give a series of explicit reductions, showing that SATISFIABILITY is reducible to each of the problems listed. Figure 1 shows the structure of the set of reductions. Each line in the figure indicates a reduction of the upper problem to the lower one.

To exhibit a reduction of a set $T \subseteq D$ to a set $T' \subseteq D'$, we specify a function $F: D \to D'$ which satisfies the conditions of Lemma 2. In each case, the reader should have little difficulty in verifying that F does satisfy these conditions.

SATISFIABILITY $\propto$ 0-1 INTEGER PROGRAMMING

$$c_{ij} = \begin{cases} 1 & \text{if } x_j \in C_i \\ -1 & \text{if } \bar{x}_j \in C_i \\ 0 & \text{otherwise} \end{cases} \qquad \begin{array}{l} i = 1,2,\ldots,p \\ j = 1,2,\ldots,n \end{array}$$

$b_i = 1 -$ (the number of complemented variables in C_i), $i = 1,2,\ldots,p$.

SATISFIABILITY $\propto$ CLIQUE

$N = \{\langle\sigma,i\rangle \mid \sigma \text{ is a literal and occurs in } C_i\}$

$A = \{\{\langle\sigma,i\rangle,\langle\delta,j\rangle\} \mid i \neq j \text{ and } \sigma \neq \bar{\delta}\}$

$k = p$, the number of clauses.

CLIQUE $\propto$ SET PACKING

Assume $N = \{1,2,\ldots,n\}$. The elements of the sets $S_1, S_2, \ldots, S_n$ are those two-element sets of nodes $\{i,j\}$ not in A.

$S_i = \{\{i,j\} \mid \{i,j\} \notin A\}$, $i = 1,2,\ldots,n$

$\ell = k$.

How do we show a problem is in **NP**?

1. Assess the size of the input instance in terms of natural parameters.
2. Define a certificate and the checking procedure for it.
3. Analyze the running time of the checking procedure, using the same natural parameters.
4. Verify that this time is polynomial in the input size.

How do we show a problem A is **NP-complete**?

1. Prove that A is in **NP**.
2. Reduce a known **NP-complete** problem to A:
   2.1 Define the reduction: how a typical instance of the known **NP-complete** problem is mapped to an instance of A.
   2.2 Prove that the reduction maps 'yes' (resp. 'no') instances of the **NP-complete** problem to a 'yes' (resp. 'no') instance of A.
   2.3 Verify that the reduction can be carried out in polynomial time.

For **NP-hardness** we do not need step 1.

Optimization problems

A decision problem has a *True* or *False* answer, whereas an optimization problem involves maximizing or minimizing a function of several parameters.

Optimization Problems

Maximize or minimize a function of the input variables.

Useful strategies for tackling **NP-hard** problems

1. Find tractable special cases which can be solved quickly.
2. Try **(meta-)heuristics** (fast, but not always correct).
3. Try exponential time algorithms better than exhaustive search.