Lecture 33:
NP-Hard and **NP**-Complete Problems

Aims:

- To describe SAT, a very important problem in complexity theory;
- To describe two more classes of problems: the **NP**-Hard and **NP**-Complete problems.

33.1. SAT

33.1.1. Descripton of SAT

- We start by looking at a decision problem that plays a major role in complexity theory. The nice thing is that it gives us another example of a problem that is in **NP**.
- Revision
  - A wff in propositional logic comprises propositional symbols (e.g. p, p_1, p_2,..., q, q_1, q_2) combined using connectives ($\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$).
  - An interpretation, $\mathcal{I}$, stipulates the truth-values of propositional symbols (e.g. p is **true**, q is **false**, etc.)
  - A wff W is satisfiable iff there is at least one interpretation that makes W **true**.
- Now here is the important problem we mentioned above:

Problem 33.1. *SAT*
Parameters: *A wff of propositional logic, W.*
Returns: *YES if W is satisfiable; NO otherwise.*

- What would SAT return for these instances?
  - $p \vee q$
  - $p \wedge \neg p$
  - $p \vee \neg p$

- $p_1 \vee (p_1 \Rightarrow ((p_2 \Rightarrow (p_3 \wedge \neg p_4) \Leftrightarrow p_5))$
- p

- (Textbook presentations of SAT are sometimes slightly different from this one. Sometimes they only allow a subset of the connectives, typically just $\wedge$, $\vee$ and $\neg$. Often they insist that the wff be in a special format called *conjunctive normal form*. This can make some proofs easier because it gives fewer connectives to consider. But none of it makes any difference to what we're doing.)

- SAT is a problem for which we know no polynomial-time algorithm.

- Yet, we have no proof that it is intractable (i.e. no proof that there cannot be a polynomial-time algorithm).

- The only algorithms we have take worst-case exponential time in n, where n is the number of propositional symbols.

- Here is an outline of one obvious exponential-time algorithm:

while there are untried interpretations
{   generate the next interpretation, $\mathcal{I}$;
      if $\mathcal{I}$ satisfies $\mathcal{W}$
      {   **return** YES;
      }
}
return NO;

- **Class Exercise:** Why does this take worst-case exponential-time?

- We can also show that SAT is in **NP**.
- Assume we have a wff W containing n distinct propositional symbols. Then here's an ND-DECAFF algorithm:

```
// The guessing part
for each distinct propositional symbol in W
{   v := choose(0, 1);
    if v = 0
    {   Assign false to the propositional symbol;
    }
    else
    {   Assign true to the propositional symbol;
    }
}
// The checking part
Evaluate W using the truth-values from above and
the truth-tables for ¬, ∧, ∨, ⇒, ⇔;
```

- Both parts of the algorithm (the guessing and the checking) take polynomial time.
- This shows that SAT is in **NP**.

33.2. NP-Hard and NP-Complete Problems

33.2.1. NP-Hard Problems

- We say that a decision problem P_i is ***NP**-hard* if *every* problem in **NP** is polynomial-time reducible to P_i.
- In symbols,

$$P_i \text{ is } \mathbf{NP}\text{-hard if, for every } P_j \in \mathbf{NP}, P_j \xrightarrow{\text{poly}} P_i.$$

- Note that this doesn't require P_i to be in **NP**.
- Highly informally, it means that P_i is 'as hard as' all the problems in **NP**.
  - If P_i can be solved in polynomial-time, then so can all problems in **NP**.
  - Equivalently, if any problem in **NP** is ever proved intractable, then P_i must also be intractable.

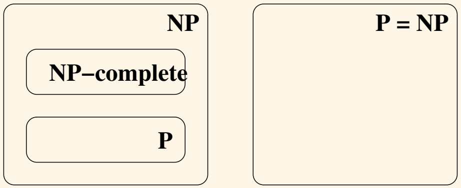
33.2.2. NP-Complete Problems

- We say that a decision problem P_i is ***NP**-complete* if
  - it is **NP**-hard and
  - it is also in the class **NP** itself.
- In symbols, P_i is **NP**-complete if P_i is **NP**-hard and $P_i \in$ **NP**
- Highly informally, it means that P_i is one of the hardest problems in **NP**.

- So the **NP**-complete problems form a set of problems that may or may not be intractable but, whether intractable or not, are all, in some sense, of equivalent complexity.
- If anyone ever shows that an **NP**-complete problem is tractable, then
  - every **NP**-complete problem is also tractable
  - indeed, every problem in **NP** is tractable

  and so $\mathbf{P} = \mathbf{NP}$.
- If anyone ever shows that an **NP**-complete problem is intractable, then
  - every **NP**-complete problem is also intractable

  and, of course, $\mathbf{P} \neq \mathbf{NP}$.
- So there are two possibilities:

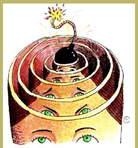


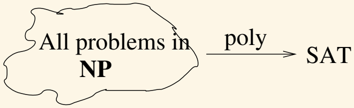
  We don't know which of these is the case.
- But this gives Computer Scientists a clear line of attack. It makes sense to focus efforts on the **NP**-complete problems: they all stand or fall together.
- So these sound like very significant problems in our theory. But how would you show that a decision problem is **NP**-complete?

- How to show a problem P_i is **NP**-complete (Method 1, from the definition)
  - First, confirm that P_i is a decision problem.
  - Then show P_i is in **NP**.
  - Then show that P_i is **NP**-hard by showing that every problem P_j in **NP** is polynomial-time reducible to P_i.
    - You wouldn't do this one by one!
    - You would try to make a general argument.

33.2.3. An NP-Complete Problem

- Definitions are all very well. But has anyone ever found an actual **NP**-complete problem? Yes!
- SAT is **NP**-complete.
- How was this proved? By method 1.
- First, SAT is a decision problem.
- Second, SAT is in **NP**.
  - We proved this earlier.
- Then it was shown SAT is **NP**-hard by showing that every problem in **NP** is polynomial-time reducible to SAT
  - This wasn't done one by one.
  - It was done by a general argument





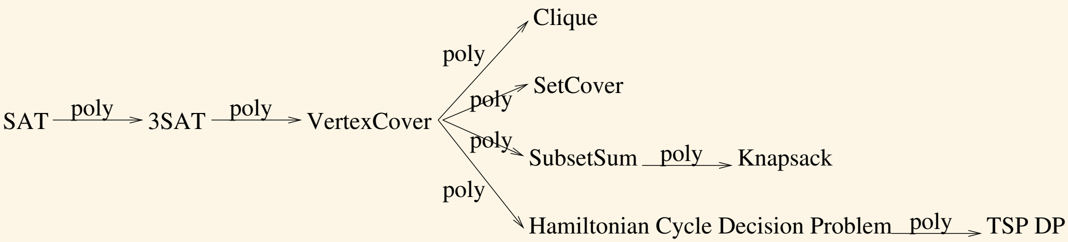
- The proof is beyond the scope of this course and the result goes by the name of the Cook-Levin Theorem

33.2.4. How to Show Other Problems are NP-Complete

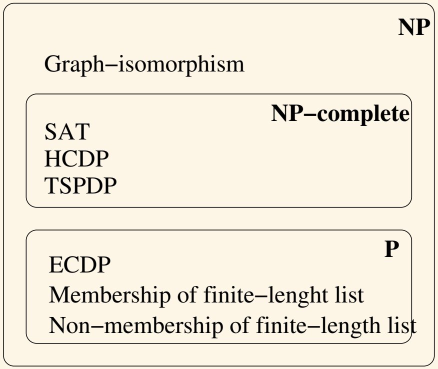
- We have one problem that is proven to be **NP**-complete, where the proof is done generically and 'from scratch'. Showing that other problems are **NP**-complete is easier.
- How to show decision problem P_i is **NP**-complete (Method 2)
  - First, confirm it is a decision problem.
  - Then show P_i is in **NP**.
  - Then show that P_i is **NP**-hard by taking just one problem P_j that is already known to be **NP**-complete and showing that $P_j \xrightarrow{\text{poly}} P_i$
- Why does the latter show P_i to be **NP**-hard?

  If P_j is **NP**-complete, then we know that P_j is **NP**-hard (by the definition of **NP**-complete), i.e. every problem in **NP** is polynomially-reducible to P_j.
  But, if every problem in **NP** is polynomially-reducible to P_j and P_j is polynomially-reducible to P_i then, by transitivity, every problem in **NP** is polynomially-reducible to P_i.

- Starting with SAT and using Method 2, numerous problems have been shown to be **NP**-complete.
- Without going into the details of the problems or the reductions themselves, here is a picture that shows a few of the polynomial-time reductions that have been found.



- Here's a picture showing some actual decision problems.



Acknowledgements:

This material owes a little to [GJ79], [GT02] and [Man89].

Clip Art (of head with bomb) licensed from the Clip Art Gallery on DiscoverySchool.com.

SAT

NP-*Hard and* **NP**-...

Module Home Page

Title Page

Page 11 of 11

Back

Full Screen

Close

Quit

References

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability.* W.H. Freeman, 1979.

[GT02] M. T. Goodrich and R. Tamassia. *Algorithm Design: Foundations, Analysis, and Internet Examples.* Wiley, 2002.

[Man89] U. Manber. *Introduction to Algorithms: A Creative Approach.* Addison Wesley, 1989.