CSC 373: Algorithm Design and Analysis
Lecture 16

Allan Borodin

February 25, 2013

Some materials are from Stephen Cook's IIT talk and Keven Wayne's slides.

Announcements and Outline

Announcements

- Lecture this Friday.
- Next assignment is due this Friday, March 1.
- Due to a seminar at 2PM (on "Algorithmic models of wireless communication" in PT266), Tuessday office hour starts at 3:10 rather than 2:30
- Please hand back your test if you have not already done do. I need to record the grade. I am still missing some tests to be recorded.
- You must keep all graded work until the term is over just in case there is some inconsistency in the grades recorded and what you have.
- Please refer to the web page for my policy on regrading.

Today's outline

- Review basic definitions of NP, reductions, and completeness
- NP-complete problems

What is NP?

- The class NP (Nondeterministic Polynomial Time)
- NP consists of all search problems for which a solution can be efficiently (i.e. in polynomial time) verified.
- More specifically, a set (or language) L is in the class NP if there is a polynomial time computable $R(x, y)$ such that for all $x \in L$, there is a certificate y such that $|y| \leq poly(|x|)$ and $R(x, y)$.

Examples in NP (besides everything in P)

- Prime Factorization
- Cracking cryptographic protocols.
- Scheduling delivery trucks, airlines, hockey matches, exams, ...

Claim

All optimization problems considered thus far have a corresponding NP decision problem.

NP-Complete Problems

- These are the hardest NP problems.
- A problem A is p-reducible to a problem B if an "oracle" (i.e. a subroutine) for B can be used to efficiently solve A.
- If A is p-reducible to B, then any efficient procedure for solving B can be turned into an efficient procedure for A.
- If A is p-reducible to B and B is in P, then A is in P.

Definition

A problem B is NP-complete if B is in NP and every problem A in NP is p-reducible to B.

Theorem

If A is NP*-complete and A is in* P*, then* $\mathrm{P} = \mathrm{NP}$.

To show $\mathrm{P} = \mathrm{NP}$ you just need to find a fast (polynomial-time) algorithm for any one NP-complete problem!!!

Reducing integer prime factorization to an NP decision problem

- As previosuly mentioned, NP is a class of decision problems whereas integer factorization is a function that takes an integer x (say in binary of decimal) and returns the unique prime factorization of x.
- And as mentioned we do abuse terminology by saying that integer factorization is an NP problem.
- The sense in which we interpret such a statement is that there is a related NP decision problem to which the factorization problem can be polyniomial time reduced; namely, consider the decision problem FACTOR $= \{(x, z) | x, z$ represent positive integers and x has a proper factor $y \leq z\}$
- Note that the decision problem COMPOSITES $= \{x | x$represents a postive integer that has a proper factor$\}$ and its complement PRIMES are known to be computable in polynomial time whereas FACTOR is believed to be difficult to compute (even in some average sense).

On the nature of polynomial time reductions

- Recalling the reduction of 3-coloring to 3-SAT, it can be seen as a very simple type of reduction.
- Namely, given an input w to the 3-coloring problem, it is transformed (in polynomial time) to say $h(w)$ such that

$$w \in \{G \mid G \text{ can be 3-colored}\} \text{ iff}$$
$$h(w) \in \{F \mid F \text{ is a satisfiable 3CNF formula}\}.$$

- This was the same kind of polynomial time reduction that showed that bipartite matching is reducible to maximum flows.

Polynomial time transformations

- We say that a language L_1 is polynomial time transformable to L_2 if there exists a polynomial time function h such that

$$w \in L_1 \text{ iff } h(w) \in L_2.$$

- The function h is called a polynomial time transformation.

Polynomial time reductions and transformations

- In practice, when we are reducing one NP problem to another NP problem, it will be a polynomial time transformation.
- We will use the same notation $\leq_p$ to denote a polynomial time reduction and polynomial time transformation.
- As we have observed before if $L_1 \leq_p L_2$ and $L_2 \in \mathrm{P}$, then $L_1 \in \mathrm{P}$.
- The contrapositive says that if $L_1 \leq_p L_2$ and $L_1 \notin \mathrm{P}$, then $L_2 \notin \mathrm{P}$.

$\leq_p$ is transitive

- An important fact that we will use to prove NP completeness of problems is that polynomial time reductions are transitive.
- That is $L_1 \leq_p L_2$ and $L_2 \leq_p L_3$ implies $L_1 \leq_p L_3$.

- The proof for transformations is easy to see. For say that $L_1 \leq_p L_2$ via g and $L_2 \leq_p L_3$ via h, then $L_1 \leq_p L_3$ via $h \circ g$;
that is, $w \in L_1$ iff $h(g(w) \in L_3$.

Polynomial reductions/transformations continued

- One fact that holds for polynomial time transformation but is believed not to hold for polynomial time reductions is the following:

NP closed under polynomial time transformation

If $L_1 \leq_p L_2$ and $L_2 \in \text{NP}$ then $L_1 \in \text{NP}$.

- The closure of NP under polynomial time transformations is also easy to see. Namely,

Suppose

- $L_2 = \{w \mid \exists y, |y| \leq q(|w|) \text{ and } R(w, y)\}$ for some polynomial time relation R and polynomial q, and
- $L_1 \leq_p L_2$ via h.

Then

$L_1 = \{x \mid \exists y', |y'| \leq q(|h(x)| \text{ and } R'(x, y')\}$ where $R'(x, y') = R(h(x), y')$

Polynomial reductions/transformations continued

- On the other hand we do not believe that NP is closed under general polynomial time reductions.
- Specifically, for general polynomial time transformations we have $\bar{L} \leq_p L$. Here $\bar{L} = \{w | w \notin L\}$ is the language complement of L.
- For example, how would you provide a short verification that a propositional formula F is *not* satisfiable? Or how would you show that a graph G *cannot* be 3-coloured?
- We do not believe that NP is closed under language complementation.
- The complement of FACTOR is an NP language and hence we do not believe that FACTOR is NP complete.
- While we will use polynomial time transformations between decision problems/languages we need to use the general polynomial reductions to say reduce a search or optimization problem to a decision problem.

So how do we show that a problem is NP complete?

- Showing that a language (i.e. decision problem) L is NP complete involves establishing two facts:

1. L is in NP
2. Showing that L is NP-hard; that is showing

$$L' \leq_p L \text{ for every } L' \in \text{NP}$$

- Usually establishing 1 is relatively easy and is done directly in terms of the definition of $L \in$ NP.
  - That is, one shows how to verify membership in L by exhibiting an appropriate certificate. (It could also be established by a polynomial time transformation to a known $L \in$ NP.)
- Establishing 2, i.e. NP-hardness of L, is usually done by reducing some known NP complete problem L' to L.

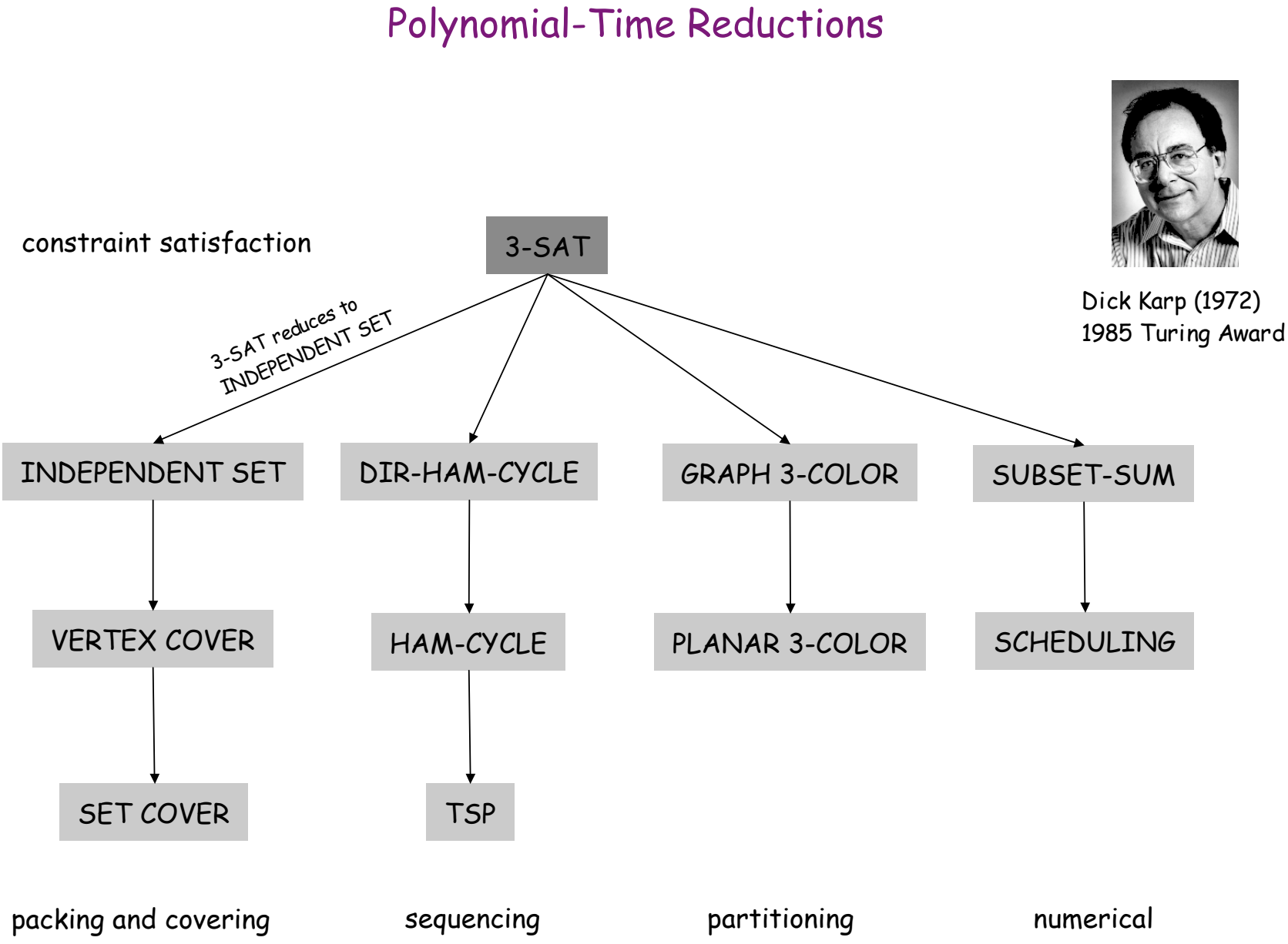
But how do we show that there are any NP complete problems?

How do we get started?

- Once we have established that there exists at least one NP complete problem then we can use polynomial time reductions and transitivity to establish that many other NP problems are NP hard.
- Following Cook's original result, we will show that SAT (and even $3SAT$) is NP complete "from first principles".

- It is easy to see that SAT is in NP.
- We will (later) show that SAT is NP hard by showing how to encode an arbitrary polynomial time (Turing) computation by a propositional formula. This will be the basis for shwoing that SAT is NP-hard.

A tree of reductions/transformations



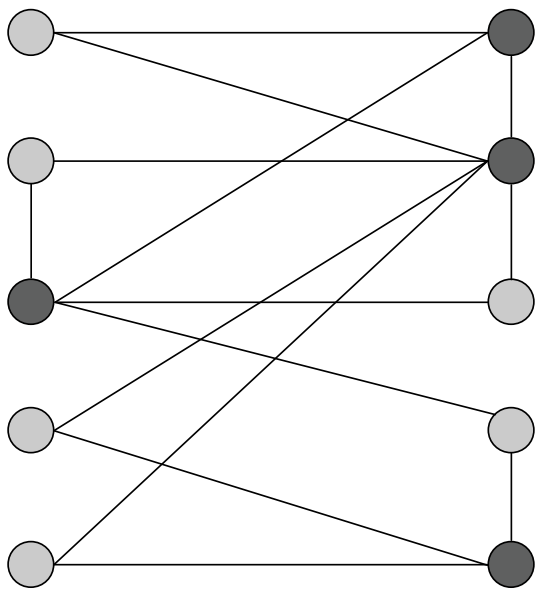
A little history of NP-completenes

- In his original 1971 seminal paper, Cook was interested in theorem proving. Stephen Cook won the Turing award in 1982
- Cook used the general notion of polynomial time reducibility which is called polynomial time Turing reducibility and sometimes called Cook reducibility.
- Cook established the NP completeness of 3SAT as well as a problem that includes CLIQUE $= \{(G, k) | G$ has a k clique $\}$.
- Independently, in the (former) Soviet Union, Leonid Levin proved an analogous result for SAT (and other problems) as a search problem.
- Following Cook's paper, Karp exhibited over 20 prominent problems that were also NP-complete.
- Karp showed that polynomial time transformations (sometimes called polynomial many to one reductions or Karp reductions) were sufficient to establish the NP completness of these problems.

Independent Set is NP complete

The independent set problem

- Given a graph $G = (V, E)$ and an integer k.
- Is there a subset of vertices $S \subseteq V$ such that $|S| \geq k$, and for each edge at most one of its endpoints is in S?

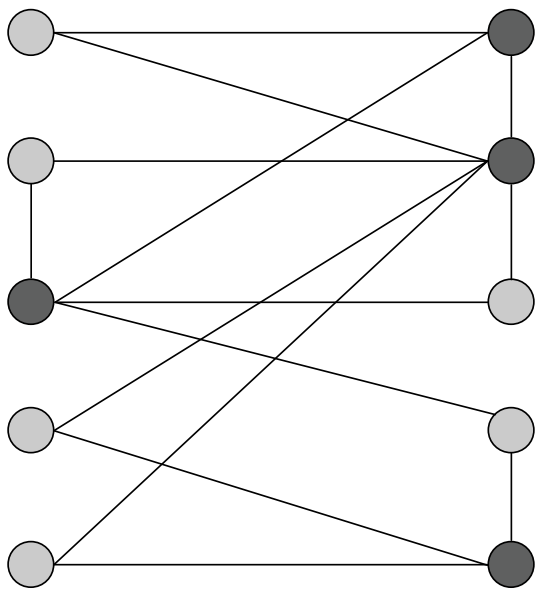


- Question: Is there an independent set of size 6?

Independent Set is NP complete

The independent set problem

- Given a graph $G = (V, E)$ and an integer k.
- Is there a subset of vertices $S \subseteq V$ such that $|S| \geq k$, and for each edge at most one of its endpoints is in S?

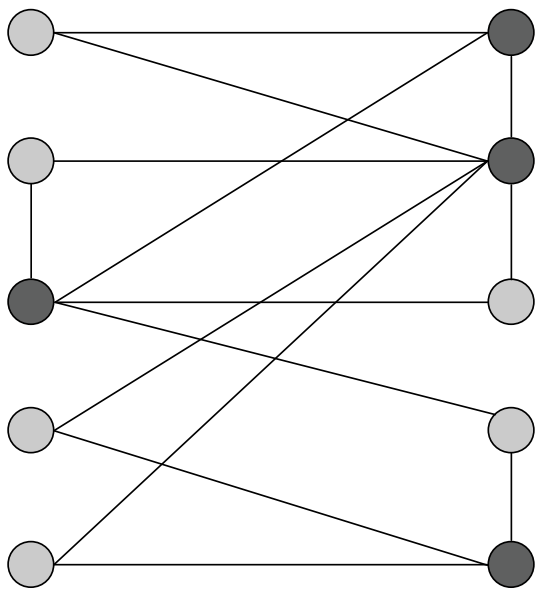


- Question: Is there an independent set of size 6? Yes.

Independent Set is NP complete

The independent set problem

- Given a graph $G = (V, E)$ and an integer k.
- Is there a subset of vertices $S \subseteq V$ such that $|S| \geq k$, and for each edge at most one of its endpoints is in S?

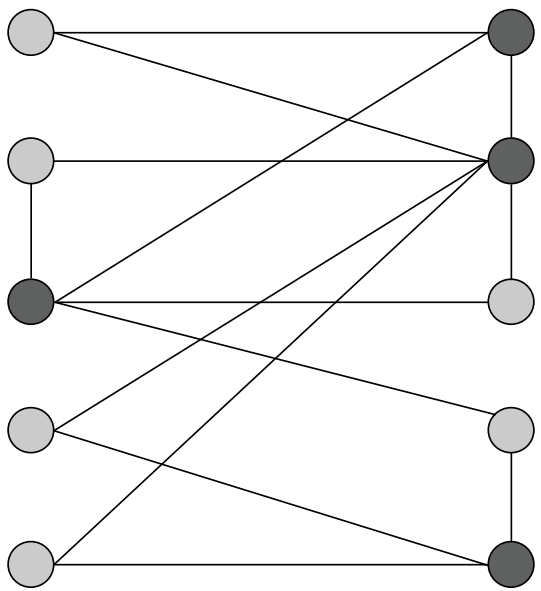


- Question: Is there an independent set of size 6? Yes.
- Question: Is there an independent set of size 7?

Independent Set is NP complete

The independent set problem

- Given a graph $G = (V, E)$ and an integer k.
- Is there a subset of vertices $S \subseteq V$ such that $|S| \geq k$, and for each edge at most one of its endpoints is in S?



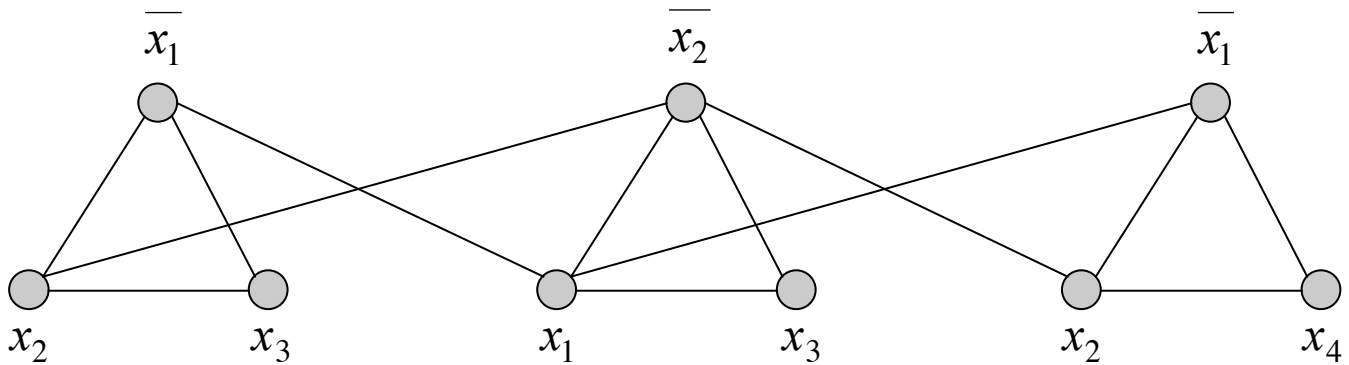
- Question: Is there an independent set of size 6? Yes.
- Question: Is there an independent set of size 7? No.

3SAT reduces to Independent Set

Claim

3SAT $\leq_p$ Independent Set

- Given an instance F of 3SAT, we construct an instance (G, k) of Independent Set that has an independent set of size k iff F is satisfiable.
- G contains 3 vertices for each clause; i.e. one for each literal.
- Connect 3 literals in a clause in a triangle.
- Connect literal to each of its negations.



$$\Phi = \left(\overline{x_1} \vee x_2 \vee x_3\right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3\right) \wedge \left(\overline{x_1} \vee x_2 \vee x_4\right)$$

Subset Sum

The independent set problem

- Given a set of integers $S = \{w_1, w_2, \ldots, w_n\}$ and an integer W.
- Is there a subset $S' \subseteq S$ that adds up to exactly W?

Example

- Given $S = \{1, 4, 16, 64, 256, 1040, 1041, 1093, 1284, 1344\}$ and $W = 3754$.
- Question: Do we have a solution?
- Answer: Yes. $1 + 16 + 64 + 256 + 1040 + 1093 + 1284 = 3754$.

3SAT reduces to Subset Sum

Claim

3SAT $\leq_p$ Subset Sum

- Given an instance F of 3SAT, we construct an instance of Subset Sum that has solution iff F is satisfiable.

$$C_1 = \bar{x} \vee y \vee z$$
$$C_2 = x \vee \bar{y} \vee z$$
$$C_3 = \bar{x} \vee \bar{y} \vee \bar{z}$$

	x	y	z	C_1	C_2	C_3
x	1	0	0	0	1	0
¬ x	1	0	0	1	0	1
y	0	1	0	1	0	0
¬ y	0	1	0	0	1	1
z	0	0	1	1	1	0
¬ z	0	0	1	0	0	1
dummies to get clause columns to sum to 4	0	0	0	1	0	0
	0	0	0	2	0	0
	0	0	0	0	1	0
	0	0	0	0	2	0
	0	0	0	0	0	1
	0	0	0	0	0	2
W	1	1	1	4	4	4