CSE 421
Algorithms

NP-Completeness
(Chapter 8)
What can we feasibly compute?

Focus so far has been to give good algorithms for specific problems (and general techniques that help do this).

Now shifting focus to problems where we think this is impossible. Sadly, there are many…
Polynomial Time
The class P

Definition: \( P = \) the set of (decision) problems solvable by computers in polynomial time, i.e.,
\[
T(n) = O(n^k) \quad \text{for some fixed } k \quad \text{(indp of input).}
\]
These problems are sometimes called tractable problems.

Examples: sorting, shortest path, MST, connectivity, RNA folding & other dyn. prog., flows & matching – i.e.: most of this qtr
(exceptions: Change-Making/Stamps, Knapsack, TSP)

(defined later)
Why “Polynomial”? 

• $n^{2000}$ is not a nice time bound
• differences among $n$, $2n$ and $n^2$ are not negligible.

*But*, simple theoretical tools don’t easily capture such differences, while exponential vs polynomial is a qualitative difference potentially more amenable to theoretical analysis.

“Problem is in P” a starting point for more detailed analysis
“Problem is not in P” may suggest that you need to shift to a more tractable variant / lower your expectations
Polynomial vs Exponential Growth

$2^{2n}$

$2^{n/10}$

$1000n^2$
Another view of Poly vs Exp

Next year’s computer will be 2x faster. If I can solve problem of size $n_0$ today, how large a problem can I solve in the same time next year?

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Increase</th>
<th>E.g. $T=10^{12}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(n)$</td>
<td>$n_0 \rightarrow 2n_0$</td>
<td>$10^{12}$</td>
</tr>
<tr>
<td>$O(n^2)$</td>
<td>$n_0 \rightarrow \sqrt{2} \cdot n_0$</td>
<td>$10^6$</td>
</tr>
<tr>
<td>$O(n^3)$</td>
<td>$n_0 \rightarrow 3\sqrt{2} \cdot n_0$</td>
<td>$10^4$</td>
</tr>
<tr>
<td>$2^n / 10$</td>
<td>$n_0 \rightarrow n_0 + 10$</td>
<td>$400$</td>
</tr>
<tr>
<td>$2^n$</td>
<td>$n_0 \rightarrow n_0 + 1$</td>
<td>$40$</td>
</tr>
</tbody>
</table>
Two Problems

How hard are they? We don’t fully know…
The Independent Set Problem

Given: a graph $G=(V,E)$ and an integer $k$

Question: is there $U \subseteq V$ with $|U| \geq k$ s.t. no pair of vertices in $U$ is joined by an edge?

What’s it good for?

E.g., if nodes = web pages, and edges join “similar” pages, then pages forming an independent set are likely to represent distinctly different topics.

E.g., if nodes = courses and edges = a student is co-enrolled, then an independent set is a set of courses whose finals could be scheduled simultaneously.

How hard is it?
Boolean Satisfiability

Boolean variables $x_1, \ldots, x_n$
  taking values in $\{0, 1\}$. 0=false, 1=true

Literals
  $x_i$ or $\neg x_i$ for $i = 1, \ldots, n$

Clause
  a logical OR of one or more literals
  e.g. $(x_1 \lor \neg x_3 \lor x_7 \lor x_{12})$

CNF formula ("conjunctive normal form")
  a logical AND of a bunch of clauses
Boolean Satisfiability

CNF formula example

\[(x_1 \lor \neg x_3 \lor x_7) \land (\neg x_1 \lor \neg x_4 \lor x_5 \lor \neg x_7)\]

If there is some assignment of 0’s and 1’s to the variables that makes it true then we say the formula is **satisfiable**

the one above is, the following isn’t

\[x_1 \land (\neg x_1 \lor x_2) \land (\neg x_2 \lor x_3) \land \neg x_3\]

**Satisfiability:** Given a CNF formula F, is it satisfiable?
Satisfiable?

( x ∨ y ∨ z ) ∧ ( ¬ x ∨ y ∨ ¬ z ) ∧
( x ∨ ¬ y ∨ z ) ∧ ( ¬ x ∨ y ∨ z ) ∧
( ¬ x ∨ ¬ y ∨ ¬ z ) ∧ ( x ∨ y ∨ z ) ∧
( x ∨ ¬ y ∨ z ) ∧ ( x ∨ y ∨ ¬ z )
Satisfiability

What’s it good for?
  Theorem provers
  Circuit validation
  Analysis of program logic
  Etc.

How hard is it?
  Don’t know fully
  Exponential time is easily possible (try all $2^n$ assignments)
  But no poly time solution is known
Reduction, I
Reductions: a useful tool

Definition: To “reduce A to B” means to solve A, given a subroutine solving B.

Example: reduce MEDIAN to SORT
   Solution: sort, then select \((n/2)\text{nd}\)

Example: reduce SORT to FIND\_MAX
   Solution: FIND\_MAX, remove it, repeat

Example: reduce MEDIAN to FIND\_MAX
   Solution: transitivity: compose solutions above.
Another Example of Reduction

reduce BIPARTITE_MATCHING to MAX_FLOW

Is there a matching of size $k$?  
Is there a flow of size $k$?

All capacities = 1
Reductions & Time

Definition: To reduce A to B means to solve A, given a subroutine solving B.

If setting up call, etc., is fast, then a fast algorithm for B implies (nearly as) an fast algorithm for A.

Contrapositive: If every algorithm for A is slow, then no algorithm for B can be fast.

“complexity of A” ≤ “complexity of B” + “complexity of reduction”
SAT and Independent Set

They are superficially different problems, but are intimately related at a deep level.
$3SAT \leq_p \text{IndpSet}$

what indp sets?
how large?
how many?
3SAT $\leq_p$ IndpSet

what indp sets?
how large?
how many?
$3\text{SAT} \leq_p \text{IndpSet}$

what indp sets?  
how large?  
how many?
$3\text{SAT} \leq_{p} \text{IndpSet}$

- What indp sets?
- How large?
- How many?

$k=3$
$3\text{SAT} \leq_p \text{IndpSet}$

$$(x_1 \lor x_2 \lor \neg x_3) \land (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor x_3)$$
$3\text{SAT} \leq_p \text{IndpSet}$

**3-SAT Instance:**
- Variables: $x_1, x_2, \ldots$
- Literals: $y_{i,j}$, $1 \leq i \leq q$, $1 \leq j \leq 3$
- Clauses: $c_i = y_{i1} \lor y_{i2} \lor y_{i3}$, $1 \leq i \leq q$
- Formula: $c = c_1 \land c_2 \land \ldots \land c_q$

**IndpSet Instance:**
- $k = q$
- $G = (V, E)$
- $V = \{ [i,j] \mid 1 \leq i \leq q, 1 \leq j \leq 3 \}$
- $E = \{ ([i,j], [k,l]) \mid i = k \text{ or } y_{ij} = \neg y_{kl} \}$
$3\text{SAT} \leq_p \text{IndpSet}$

$$(x_1 \lor x_2 \lor \neg x_3) \land (x_1 \lor \neg x_2 \lor \neg x_3)$$

$k=2$
$3\text{SAT} \leq_p \text{IndpSet}$
Correctness of “3SAT ≤ₚ IndpSet”

Summary of reduction function f: Given formula, make graph G with one group per clause, one node per literal. Connect each to all nodes in same group; connect all complementary literal pairs (x, ¬x). Output graph G plus integer k = number of clauses. Note: f does not know whether formula is satisfiable or not; does not know if G has k-IndpSet; does not try to find satisfying assignment or set.

Correctness:
• Show f poly time computable: A key point is that graph size is polynomial in formula size; mapping basically straightforward.
• Show c in 3-SAT iff f(c)=(G,k) in IndpSet:
  (⇒) Given an assignment satisfying c, pick one true literal per clause. Add corresponding node of each triangle to set. Show it is an IndpSet: 1 per triangle never conflicts w/ another in same triangle; only true literals (but perhaps not all true literals) picked, so not both ends of any (x, ¬x) edge.
  (⇐) Given a k-Independent Set in G, selected labels define a valid (perhaps partial) truth assignment since no (x, ¬x) pair picked. It satisfies c since there is one selected node in each clause triangle (else some other clause triangle has > 1 selected node, hence not an independent set.)
Utility of “3SAT $\leq_p$ IndpSet”

Suppose we had a fast algorithm for IndpSet, then we could get a fast algorithm for 3SAT:

Given 3-CNF formula $w$, build Independent Set instance $y = f(w)$ as above, run the fast IS alg on $y$; say “YES, $w$ is satisfiable” iff IS alg says “YES, $y$ has a Independent Set of the given size”

On the other hand, suppose no fast alg is possible for 3SAT, then we know none is possible for Independent Set either.
“3SAT \leq_p \text{IndpSet}” Retrospective

Previous slides: two suppositions
Somewhat clumsy to have to state things that way.
Alternative: abstract out the key elements, give it a name ("polynomial time mapping reduction"), then properties like the above always hold.
Reduction, II

Polynomial time “mapping” reduction
Polynomial-Time Reductions

Definition: Let $A$ and $B$ be two decision problems. $A$ is \textit{polynomially (mapping) reducible} to $B$ ($A \leq_p B$) if there exists a polynomial-time algorithm $f$ that converts each instance $x$ of problem $A$ to an instance $f(x)$ of $B$ such that:

$$x \text{ is a YES instance of } A \iff f(x) \text{ is a YES instance of } B$$

$$x \in A \iff f(x) \in B$$
Polynomial-Time Reductions (cont.)

**Defn:** $A \leq_p B$ “$A$ is polynomial-time reducible to $B$,” iff there is a polynomial-time computable function $f$ such that: $x \in A \iff f(x) \in B$

“complexity of $A$” $\leq$ “complexity of $B$” + “complexity of $f$”

**Theorem:**

1. $A \leq_p B$ and $B \in P \implies A \in P$
2. $A \leq_p B$ and $A \notin P \implies B \notin P$
3. $A \leq_p B$ and $B \leq_p C \implies A \leq_p C$ (transitivity)
Another Example Reduction

SAT to Subset Sum (Knapsack)
Subset-Sum, AKA Knapsack

KNAP = \{ (w_1, w_2, \ldots, w_n, C) \mid \text{a subset of the } w_i \text{ sums to } C \}

w_i’s and C encoded in radix r ≥ 2. (Decimal used in following example.)

Theorem: 3-SAT ≤_P KNAP

Pf: given formula with p variables & q clauses, build KNAP instance with 2(p+q) w_i’s, each with (p+q) decimal digits. See examples below.
### 3-SAT $\leq_p$ KNAP

**Formula:** $(x \lor y \lor z)$

<table>
<thead>
<tr>
<th>Literals</th>
<th>Variables</th>
<th>Clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1$ ($x$)</td>
<td>$1$</td>
<td>$1$</td>
</tr>
<tr>
<td>$w_2$ ($\neg x$)</td>
<td>$1$</td>
<td>$0$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Slack</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_7$ ($s_{11}$)</td>
<td></td>
<td>$1$</td>
</tr>
<tr>
<td>$w_8$ ($s_{12}$)</td>
<td></td>
<td>$1$</td>
</tr>
</tbody>
</table>

**What/How Many Satisfying Assignments?**

**What/How Many KNAP solutions?**

| | $l$ | $3$ |
### 3-SAT \(\leq_p\) KNAP

**Formula:** \((x \lor y \lor z) \land (\neg x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z)\)

<table>
<thead>
<tr>
<th>Variables</th>
<th>Clauses</th>
</tr>
</thead>
</table>
| \(x\)    | \((x\) \((\neg x\) ) \)

<table>
<thead>
<tr>
<th>Literals</th>
<th>Variables</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(w_1) ((x))</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>(w_2) ((\neg x))</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Slack</th>
<th>Variables</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(w_7) ((s_{11}))</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(w_8) ((s_{12}))</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(w_9) ((s_{21}))</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(w_{10}) ((s_{22}))</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**What/How Many Satisfying Assignments?**

**What/How Many KNAP solutions?**
### 3-SAT $\leq_p$ KNAP

Formula: $(x \lor y \lor z)$

<table>
<thead>
<tr>
<th>Literals</th>
<th>Variables</th>
<th>Clauses</th>
<th>What/How Many Satisfying Assignments?</th>
<th>What/How Many KNAP solutions?</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1 (x)$</td>
<td>$x$</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$w_2 (\neg x)$</td>
<td>$x$</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$w_3 (y)$</td>
<td>$y$</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$w_4 (\neg y)$</td>
<td>$y$</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$w_5 (z)$</td>
<td>$z$</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$w_6 (\neg z)$</td>
<td>$z$</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$w_7 (s_{11})$</td>
<td>Slack</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$w_8 (s_{12})$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| $C$ | 1 | 1 | 1 | 3 |
### 3-SAT $\leq_p$ KNAP

Formula: $(x \lor y \lor z) \land (\neg x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z)$

<table>
<thead>
<tr>
<th>Literals</th>
<th>Variables</th>
<th>Clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1$ ($x$)</td>
<td>$1 \ 0 \ 0$</td>
<td>$1 \ 0 \ 0$</td>
</tr>
<tr>
<td>$w_2$ ($\neg x$)</td>
<td>$1 \ 0 \ 0$</td>
<td>$0 \ 1 \ 1$</td>
</tr>
<tr>
<td>$w_3$ ($y$)</td>
<td>$1 \ 0$</td>
<td>$1 \ 1 \ 0$</td>
</tr>
<tr>
<td>$w_4$ ($\neg y$)</td>
<td>$1 \ 0$</td>
<td>$0 \ 0 \ 1$</td>
</tr>
<tr>
<td>$w_5$ ($z$)</td>
<td>$1 \ 1$</td>
<td>$1 \ 0 \ 1$</td>
</tr>
<tr>
<td>$w_6$ ($\neg z$)</td>
<td>$1 \ 0$</td>
<td>$1 \ 0 \ 0$</td>
</tr>
<tr>
<td>Slack</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$w_7$ ($s_{11}$)</td>
<td></td>
<td>$1 \ 0 \ 0$</td>
</tr>
<tr>
<td>$w_8$ ($s_{12}$)</td>
<td></td>
<td>$1 \ 0 \ 0$</td>
</tr>
<tr>
<td>$w_9$ ($s_{21}$)</td>
<td></td>
<td>$1 \ 0 \ 0$</td>
</tr>
<tr>
<td>$w_{10}$ ($s_{22}$)</td>
<td></td>
<td>$1 \ 0 \ 0$</td>
</tr>
<tr>
<td>$w_{11}$ ($s_{31}$)</td>
<td></td>
<td>$1 \ 0 \ 0$</td>
</tr>
<tr>
<td>$w_{12}$ ($s_{32}$)</td>
<td></td>
<td>$1 \ 0 \ 0$</td>
</tr>
<tr>
<td>$C$</td>
<td>$1 \ 1 \ 1$</td>
<td>$3 \ 3 \ 3$</td>
</tr>
</tbody>
</table>

What/How Many Satisfying Assignments/KNAP solutions?
3-SAT $\leq_p$ KNAP

3-SAT Instance:
- Variables: $x_1, x_2, \ldots, x_p$
- Literals: $y_{i,j}$, $1 \leq i \leq q$, $1 \leq j \leq 3$
- Clauses: $c_i = y_{i1} \lor y_{i2} \lor y_{i3}$, $1 \leq i \leq q$
- Formula: $c = c_1 \land c_2 \land \ldots \land c_q$

KNAP Instance:
- 2($p+q$) wi’s, each with ($p+q$) decimal digits, mostly 0
- For the 2$p$ “literal” weights, a single 1 in H.O. $p$ digits marks which variable; 1’s in L.O. $q$ digits mark each clause containing that literal.
- Two “slacks” per clause; single 1 marks the clause.
- Knapsack Capacity $C = 11..133..3$ ($p$ 1’s, $q$ 3’s)
Correctness

Poly time for reduction is routine; details omitted. Note that it does not look at satisfying assignment(s), if any, nor at subset sums (but the problem instance it builds captures one via the other...)

If formula is satisfiable, select the literal weights corresponding to the true literals in a satisfying assignment. If that assignment satisfies k literals in a clause, also select (3 - k) of the “slack” weights for that clause. Total = C.

Conversely, suppose KNAP instance has a solution. Columns are decoupled since \( \leq 5 \) one’s per column, so no “carries” in sum (recall – weights are decimal). Since H.O. \( p \) digits of \( C \) are 1, exactly one of each pair of literal weights included in the subset, so it defines a valid assignment. Since L.O. \( q \) digits of \( C \) are 3, but at most 2 “slack” weights contribute to each, at least one of the selected literal weights must be 1 in that clause, hence the assignment satisfies the formula.
Decision vs Search Problems
The Clique Problem

Given: a graph $G=(V,E)$ and an integer $k$

Question: is there a subset $U$ of $V$ with $|U| \geq k$ such that every pair of vertices in $U$ is joined by an edge.

E.g., if nodes are web pages, and edges join “similar” pages, then pages forming a clique are likely to be about the same topic.
Problem Types

A *clique* in an undirected graph \(G=(V,E)\) is a subset \(U\) of \(V\) such that every pair of vertices in \(U\) is joined by an edge.

E.g., mutual friends on Facebook, genes that vary together

An *optimization* problem: *How large* is the largest clique in \(G\)

A *search* problem: *Find* the/a largest clique in \(G\)

A *search* problem: Given \(G\) and integer \(k\), *find* a \(k\)-clique in \(G\)

A *decision* problem: Given \(G\) and \(k\), *is there* a \(k\)-clique in \(G\)

A *verification* problem: Given \(G\), \(k\), \(U\), *is \(U\)* a \(k\)-clique in \(G\)
Some Convenient Technicalities

“Problem” – the general case
Ex: The Clique Problem: Given a graph $G$ and an integer $k$, does $G$ contain a $k$-clique?

“Problem Instance” – the specific cases
Ex: Does contain a 4-clique? (no)
Ex: Does contain a 3-clique? (yes)

Problems as Sets of “Yes” Instances
Ex: $\text{CLIQUE} = \{ (G,k) \mid G \text{ contains a } k\text{-clique} \}$
E.g., $(\text{ }, 4) \notin \text{CLIQUE}$
E.g., $(\text{ }, 3) \in \text{CLIQUE}$
Beyond P
Beyond P?

There are many natural, practical problems for which we don’t know any polynomial-time algorithms:

- e.g. SAT, IndpSet, CLIQUE, KNAP, TSP, …

Lack of imagination or intrinsic barrier?
NP
Roadmap

Not every problem is easy (in P)

Exponential time is bad

Worse things happen, too

There is a very commonly-seen class of problems, called NP, that appear to require exponential time (but unproven)
Review: Some Problems

Quadratic Diophantine Equations
Clique
Independent Set
Euler Tour
Hamilton Tour
TSP
3-Coloring
Partition
Satisfiability
Short Paths
Long Paths

All of the form: Given input X, is there a Y with property Z? Furthermore, if I had a purported Y, I could quickly test whether it had property Z.
Common property of these problems: Discrete Exponential Search
Loosely—find a needle in a haystack

“Answer” to a decision problem is literally just yes/no, but there’s always a somewhat more elaborate “solution” (aka “hint” or “certificate”; what the search version would report) that transparently‡ justifies each “yes” instance (and only those) – but it’s buried in an exponentially large search space of potential solutions.

‡Transparently = verifiable in polynomial time
Defining NP: Informally

NP is the set of decision problems where

There is a closely related search problem such that
For all “Yes” instances of the decision version
If I could guess a solution to the search problem
You could “check” my guess quickly (P-time)

But

Your check wouldn’t be fooled by anything I say
about a “No” instance
Defining NP: formally

A decision problem $L$ is in \textit{NP} iff there is a polynomial time procedure $v(\cdot,\cdot)$, (the “verifier”) and an integer $k$ such that

- for every $x \in L$ there is a “hint” $h$ with $|h| \leq |x|^k$ such that $v(x,h) = \text{YES}$

and

- for every $x \not\in L$ there is no hint $h$ with $|h| \leq |x|^k$ such that $v(x,h) = \text{YES}$

(“Hints,” sometimes called “certificates,” or “witnesses”, are just strings. Think of them as exactly what the search version would output.)

Note 1: a problem is “in NP” if it can be posed as an exponential search problem, even if there may be other ways to solve it.

Note 2: his definition is not quickly actionable without a way to find $h$. 
Example: Clique

“Is there a $k$-clique in this graph?”
any subset of $k$ vertices might be a clique
there are many such subsets, but I only need to find one
if I knew where it was, I could describe it succinctly, e.g.
“look at vertices 2, 3, 17, 42, ...”,
I’d know one if I saw one: “yes, there are edges between
2 & 3, 2 & 17,... so it’s a $k$-clique”
this can be quickly checked
And if there is no $k$-clique, I wouldn’t be fooled
by a statement like “look at vertices 2, 3, 17, 42, ...”
More Formally: CLIQUE is in NP

procedure \( v(x, h) \)

if

\( x \) is a well-formed representation of a graph \( G = (V, E) \) and an integer \( k \),

and

\( h \) is a well-formed representation of a \( k \)-vertex subset \( U \) of \( V \),

and

\( U \) is a clique in \( G \),

then output “YES”

else output “I’m unconvinced”
Is it correct?

For every $x = (G,k)$ such that $G$ contains a $k$-clique, there is a hint $h$ that will cause $v(x,h)$ to say YES, namely $h$ = a list of the vertices in such a $k$-clique and

No hint can fool $v$ into saying yes if either $x$ isn’t well-formed (the uninteresting case) or if $x = (G,k)$ but $G$ does not have any cliques of size $k$ (the interesting case)

And $|h| < |x|$ and $v(x,h)$ takes time $\sim (|x|+|h|)^2$
“Is there a satisfying assignment for this Boolean formula?”

any assignment might work
there are lots of them
I only need one
if I had one I could describe it succinctly, e.g., “\(x_1=T, x_2=F, \ldots, x_n=T\)"
I’d know one if I saw one: “yes, plugging that in, I see formula = T…”
and this can be quickly checked
And if the formula is unsatisfiable, I wouldn’t be fooled by , “\(x_1=T, x_2=F, \ldots, x_n=F\)”
More Formally: SAT ∈ NP

Hint: the satisfying assignment A

Verifier: v(C, A) = syntax(C, A) && satisfies(C, A)

Syntax: True iff C is a well-formed CNF formula & A is a truth-assignment to its variables
Satisfies: plug A into C; check that it evaluates to True

Correctness:
If C is satisfiable, it has some satisfying assignment A, and we’ll recognize it
If C is unsatisfiable, it doesn’t, and we won’t be fooled

Analysis: |A| < |C|, and time for v(C,A) ~ linear in |C|+|A|_57
IndpSet is in NP

procedure v(x,h)
    if
        x is a well-formed representation of a graph G = (V, E) and an integer k,
        and
        h is a well-formed representation of a k-vertex subset U of V,
        and
        U is an Indp Set in G,
    then output “YES”
    else output “I’m unconvinced”

Important note: this answer does NOT mean x ∉ IndpSet; just means this h isn’t a k-IndpSet (but some other might be).
Is it correct?

For every \( x = (G,k) \) such that \( G \) contains a \( k \)-IndpSet, there is a hint \( h \) that will cause \( v(x,h) \) to say YES, namely \( h \) = a list of the vertices in such a set and

No hint can fool \( v \) into saying yes if either \( x \) isn’t well-formed (the uninteresting case) or if \( x = (G,k) \) but \( G \) does not have any Indp Set of size \( k \) (the interesting case)

And \( |h| < |x| \) and \( v(x,h) \) takes time \( \sim (|x|+|h|)^2 \)
Keys to showing that a problem is in NP

What’s the output? (must be YES/NO)
What’s the input? Which are YES?
For every given YES input, is there a hint that would help, i.e. allow verification in polynomial time? Is it polynomial length?
   OK if some inputs need no hint
For any given NO input, is there a hint that would trick you?
Two Final Points About “Hints”

1. Hints/verifiers aren’t unique. The “… there is a …” framework often suggests their form, but many possibilities

   “is there a clique” could be verified from its vertices, or its edges, or all but 3 of each, or all non-vertices, or… Details of the hint string and the verifier and its time bound shift, but same bottom line

2. In NP doesn’t prove its hard

   “Short Path” or “Small Spanning Tree” or “Large Flow” can be formulated as “…there is a…,” but, due to very special structure of these problems, we can quickly find the solution even without a hint. The mystery is whether that’s possible for the other problems, too.
Contrast: problems not in \textbf{NP} (probably)

Rather than “there is a…” maybe it’s
“\textit{no}…” or “\textit{for all}…” or “\textit{the smallest/largest}…”

E.g.

\textbf{UNSAT}: “\textit{no} assignment satisfies formula,” or
“\textit{for all} assignments, formula is false”

Or

\textbf{NOCLIQUE}: “\textit{every} subset of \textit{k} vertices is not a \textit{k}-clique”
\textbf{MAXCLIQUE}: “the largest clique has size \textit{k}”

Unlikely that a single, short hint is sufficiently informative to allow poly time verification of properties like these (but this is also an important open problem).
NP-completeness
NP-Completeness

Definition: Problem B is *NP-hard* if every problem in NP is polynomially reducible to B.

Definition: Problem B is *NP-complete* if:

1. B belongs to NP, and
2. B is NP-hard.
NP-completeness (cont.)

Thousands of important problems have been shown to be NP-complete.

The general belief is that there is no efficient algorithm for any NP-complete problem, but no proof of that belief is known.

Examples: SAT, clique, vertex cover, IndpSet, Ham tour, TSP, bin packing... Basically, everything we’ve seen that’s in NP but not known to be in P
Proving a problem is NP-complete

Technically, for condition (2) we have to show that every problem in NP is reducible to B. (Sounds like a lot of work!)
For the very first NP-complete problem (SAT) this had to be proved directly.
However, once we have one NP-complete problem, then we don’t have to do this every time.
Why? Transitivity of $\leq_p$. 
Alt way to prove NP-completeness

Lemma: Problem B is NP-complete iff:

(1) B belongs to NP, and

(2’) A is polynomial-time reducible to B, for some problem A that is NP-complete.

That is, to show NP-completeness of a new problem B in NP, it suffices to show that SAT or any other NP-complete problem is polynomial-time reducible to B.
Ex: IndpSet is NP-complete

3-SAT is NP-complete (S. Cook; see below)
3-SAT \leq_p \text{IndpSet} \quad \text{we showed these earlier}
IndpSet is in NP
Therefore IndpSet is also NP-complete

So, poly-time algorithm for IndpSet would give poly-time algs for everything in NP

Ditto for KNAP, 3COLOR, …
Cook’s Theorem

SAT is NP-Complete
“NP-completeness”

Cool concept, but are there any such problems?

Yes!

Cook’s theorem: SAT is NP-complete
Why is SAT NP-complete?

Cook’s proof is somewhat involved. I’ll sketch it below. But its essence is not so hard to grasp:

Generic “NP” problems: expo. search—
is there a poly size “solution,” verifiable by computer in poly time

“SAT”: is there a poly size assignment (the hint) satisfying the formula (the verifier)

Encode “solution” using Boolean variables. SAT mimics “is there a solution” via “is there an assignment”. The “verifier” runs on a digital computer, and digital computers just do Boolean logic. “SAT” can mimic that, too, hence can verify that the assignment actually encodes a solution.
Examples

Again, Cook’s theorem does this for *generic* NP problems, but you can get the flavor from a few specific examples
3-Coloring $\leq_p$ SAT

Given $G = (V, E)$

$\forall$ i in $V$, variables $r_i, g_i, b_i$ encode color of i

$\wedge_{i \in V} [(r_i \lor g_i \lor b_i) \land (\neg r_i \lor \neg g_i) \land (\neg g_i \lor \neg b_i) \land (\neg b_i \lor \neg r_i)] \land \wedge_{(i,j) \in E} [(\neg r_i \lor \neg r_j) \land (\neg g_i \lor \neg g_j) \land (\neg b_i \lor \neg b_j)]$

Equivalently:

$(\neg (r_i \land g_i)) \land (\neg (g_i \land b_i)) \land (\neg (b_i \land r_i)) \land \wedge_{(i,j) \in E} [(r_i \Rightarrow \neg r_j) \land (g_i \Rightarrow \neg g_j) \land (b_i \Rightarrow \neg b_j)]$

adj nodes $\Leftrightarrow$ diff colors
no node gets 2
ev every node gets a color
Independent Set $\leq_p$ SAT

Given $G = (V, E)$ and $k$

$\forall$ $i$ in $V$, variable $x_i$ encodes inclusion of $i$ in IS

$$\land_{(i,j) \in E} (\neg x_i \lor \neg x_j) \land \text{“number of True } x_i \text{ is } \geq k”$$

every edge has one end
or other not in IS
(no edge connects 2 in IS)

possible in 3 CNF, but technically
messy, so details omitted;
basically, count 1’s
Vertex cover $\leq_p$ SAT

Given $G = (V, E)$ and $k$

$\forall$ $i$ in $V$, variable $x_i$ encodes inclusion of $i$ in cover

$\wedge_{(i,j) \in E} (x_i \lor x_j) \wedge$ “number of True $x_i$ is $\leq k$”

every edge covered by one end or other

possible in 3 CNF, but technically messy; basically, count 1’s
Hamilton Circuit $\leq_p$ SAT

Given $G = (V, E)$ [encoded, e.g.: $e_{ij} = 1 \iff$ edge $(i,j)$]

\forall i,j \in V, \text{ variables } x_{ij}, \text{ encode } \text{“j follows i in the tour”} \quad \text{hint}

$$\wedge_{(i,j)} (x_{ij} \Rightarrow e_{ij}) \wedge \text{“it’s a permutation”} \wedge \text{“cycle length = n”}$$

the path follows actual edges

every row/column has exactly 1 one bit

$X^n = 1$, no smaller power $k$ has $X^{k_{ii}} = 1$
Perfect Matching $\leq_p$ SAT

Given $G = (V, E)$ [encoded, e.g.: $e_{ij} = 1 \iff$ edge $(i,j)$]

$\forall i < j$ in $V$, variable $x_{ij}$, encodes “edge $i,j$ is in matching”

$$\left( \land_{i < j} (x_{ij} \Rightarrow e_{ij}) \right) \land \left( \land_{i < j < k} (x_{ij} \Rightarrow \neg x_{ik}) \right) \land \left( \land_i (\lor_j x_{ij}) \right)$$

- matching edges are actual edges
- it’s a matching: if edge $(i,j)$ included, then $(i,k)$ excluded
- all vertices are matched
Cook’s Theorem

Every problem in NP is reducible to SAT

Idea of proof is extension of above examples, but done in a general way, based on the definition of NP – show how the SAT formula can simulate whatever (polynomial time) computation the verifier does.

Cook proved it directly, but easier to see via an intermediate problem – Satisfiability of Circuits rather than Formulas
Boolean Circuits

Directed acyclic graph (yes, “circuit” is a misnomer…)
Vertices = Boolean logic gates (\(\land, \lor, \neg, \ldots\)) + inputs
Multiple input bits (\(x_1, x_2, \ldots\))
Single output bit (\(w\))
Gate values as expected (e.g., propagate vals by depth to \(x_i\)’s)
Boolean Circuits and Complexity

Two Problems:

- Circuit Value: given a circuit *and an assignment* of values to its inputs, is its output = 1?
- Circuit SAT: given a circuit, *is there* an assignment of values to its inputs such that output = 1?

Complexity:

- Circuit Value Problem is in P
- Circuit SAT Problem is in NP

Given implementation of computers via Boolean circuits, it may be unsurprising that they are complete in P/NP, resp.
Detailed Logic Diagram, Intelorola Pentathlon® 66000

Registers/Latches/Memory

Combinational Logic, Large Rat’s Nest of

Really, Really, Fast Clock
P Is Reducible To The Circuit Value Problem

Registers/Latches/Memory

Combinational Logic, Large Rat’s Nest of

Really, Really, Fast Clock

Answer yes?

0 1 0 … Input … 0 1 1

Combinational Logic

Combinational Logic

Combinational Logic
NP Is Reducible To The Circuit Satisfiability Problem

Registers/Latches/Memory

Combinational Logic, Large Rat’s Nest of

Really, Really, Fast Clock

0 1 ...Input... 0 1

??..Hint..??

Combinational Logic

The Verifier Algorithm

Combinational Logic

Answer yes?
Correctness of $\text{NP} \leq_p \text{CircuitSAT}$

Fix an arbitrary NP-problem, a verifier alg $V(x,h)$ for it, and a bound $n^k$ on hint length/run time of $V$, show:

1) In poly time, given $x$, can output a circuit $C$ as above,
2) $\exists$ h s.t. $V(x,h)=\text{"yes"} \Rightarrow C$ is satisfiable (namely by h), and
3) $C$ is satisfiable (say, by h) $\Rightarrow \exists$ h s.t. $V(x,h)=\text{"yes"}$

1) is perhaps very tedious, but mechanical—you are “compiling” the verifier’s code into hardware (just enough hardware to handle all inputs of length $|x|$)

2) & 3) exploit the fact that $C$ simulates $V$, with $C$’s “hint bit” inputs exactly corresponding to $V$’s input h.
Circuit-SAT \leq_p 3-SAT

\((w_1 \leftrightarrow (x_1 \land x_2)) \land (w_2 \leftrightarrow (\neg w_1)) \land (w_3 \leftrightarrow (w_2 \lor x_1)) \land w_3\)

Replace with 3-CNF Equivalent:

\[
f(\text{Circuit-SAT}) = (x_1 \lor x_2 \lor \neg w_1) \land (x_1 \lor \neg x_2 \lor \neg w_1) \land (\neg x_1 \lor x_2 \lor \neg w_1) \land (\neg x_1 \lor \neg x_2 \lor w_1) \ldots
\]

Q. Why build truth table clause-by-clause vs whole formula? A: So \(n \ast 2^3\) vs \(2^n\) rows
Correctness of “Circuit-SAT \( \leq_p 3\text{-SAT} \)”

Summary of reduction function \( f \): Given circuit, add variable for every gate’s value, build clause for each gate, satisfiable iff gate value variable is appropriate logical function of its input variables, convert each to CNF via standard truth-table construction. Output conjunction of all, plus output variable. *Note: as usual, does not know whether circuit or formula are satisfiable or not; does not try to find satisfying assignment.*

Correctness:

Show \( f \) is poly time computable: A key point is that formula size is linear in circuit size; mapping basically straightforward; details omitted.

Show \( c \) in Circuit-SAT iff \( f(c) \) in SAT:

\( \Rightarrow \) Given an assignment to \( x_i \)’s satisfying \( c \), extend it to \( w_i \)’s by evaluating the circuit on \( x_i \)’s gate by gate. Show this satisfies \( f(c) \).

\( \Leftarrow \) Given an assignment to \( x_i \)’s & \( w_i \)’s satisfying \( f(c) \), show \( x_i \)’s satisfy \( c \) (with gate values given by \( w_i \)’s).

Thus, 3-SAT is NP-complete.
Relating P to NP
NP = Polynomial-time verifiable

P = Polynomial-time solvable

P ⊆ NP: “verifier” is just the P-time alg; ignore “hint”
Solving NP problems without hints

The most obvious algorithm for most of these problems is brute force:
try all possible hints; check each one to see if it works.

Exponential time:
\[2^n\] truth assignments for \(n\) variables
\[n!\] possible TSP tours of \(n\) vertices
\[\binom{n}{k}\] possible \(k\) element subsets of \(n\) vertices, perhaps \(k = \log n\) or \(n/3\)

…and to date, every alg, even much less-obvious ones, are slow, too
P vs NP vs Exponential Time

Theorem: Every problem in NP can be solved (deterministically) in exponential time

Proof: “hints” are only $n^k$ long; try all $2^{n^k}$ possibilities, say, by backtracking. If any succeed, answer YES; if all fail, answer NO.
Every problem in P is in NP
one doesn’t even need a hint for problems in P so just ignore any hint you are given

Every problem in NP is in exponential time

I.e., $P \subseteq NP \subseteq Exp$

We know $P \neq Exp$, so either $P \neq NP$, or $NP \neq Exp$ (most likely both)
Does P = NP?

This is the big open question!
To show that P = NP, we have to show that every problem that belongs to NP can be solved by a polynomial time deterministic algorithm.
Would be very cool, but no one has shown this yet.
(And it seems unlikely to be true.)
Polynomial Time Reduction, III
Two definitions of “$A \leq_p B$”

Book uses general definition: “could solve A in poly time, if I had a poly time subroutine for B.”

Examples on previous slides are special case where you only call the subroutine once, and must report its answer.

This special case is used in ~98% of all reductions

Largely irrelevant for this course, but if you seem to need 1st defn, e.g. on HW, fine, but there’s perhaps a simpler way…
Example of the difference

\[ \text{CLIQUE} = \{ (G,k) \mid \text{G has a k-clique} \} \]
\[ \text{MAXCLIQUE} = \{ (G,k) \mid \text{G’s largest clique is size k} \} \]

Q: is MAXCLIQUE $\in$ NP?

A: probably not; a hint might give you a k-clique (& you could check it), but what “hint” would also convince you of absence of \( (k+1) \)-cliques?

Theorem: CLIQUE $\leq_{p}^{\text{Cook}}$ MAXCLIQUE, so later is NP-Hard

Pf: Ptime alg for CLIQUE, given hypothetical ptime subr for MAXCLIQUE:

\[
\text{CLIQUE}_\text{Alg}(G,k):
\]
\[
\text{for j=k,...,|G| } \{
\]
\[
\text{if MAXCLIQUE}_\text{Subr}(G,j) \text{ says “yes”}
\]
\[
\text{then return “Yes, (G,k) } \in \text{ CLIQUE”}
\]
\[
\}
\]
\[
\text{return “No, (G,k) } \notin \text{ CLIQUE”}
\]

Exercise: show MAXCLIQUE $\leq_{p}^{\text{Cook}}$ CLIQUE
More Reductions

SAT to Coloring
NP-complete problem: 3-Coloring

Input: An undirected graph $G=(V,E)$.
Output: True iff there is an assignment of at most 3 colors to the vertices in $G$ such that no two adjacent vertices have the same color.

Example:

In NP? Exercise
A 3-Coloring Gadget:

In what ways can this be 3-colored?
A 3-Coloring Gadget: “Sort of an OR gate“

if output is T, some input must be T

Exercise: find all colorings of 5 nodes

NB: this is \textit{not} the same gadget as used in KT 8.7
3SAT $\leq_p$ 3Color

3-SAT Instance:
- Variables: $x_1, x_2, \ldots$
- Literals: $y_{i,j}, 1 \leq i \leq q, 1 \leq j \leq 3$
- Clauses: $c_i = y_{i1} \lor y_{i2} \lor y_{i3}, 1 \leq i \leq q$
- Formula: $c = c_1 \land c_2 \land \ldots \land c_q$

3Color Instance:
- $G = (V, E)$
- 6$q + 2n + 3$ vertices
- 13$q + 3n + 3$ edges
- (See Example for details)
3SAT $\leq_p$ 3Color Example

$(x_1 \lor \neg x_1 \lor \neg x_1) \land (\neg x_1 \lor x_2 \lor \neg x_2)$

6 $q + 2$ $n + 3$ vertices 13 $q + 3$ $n + 3$ edges
Correctness of “3SAT $\leq_p$ 3Coloring”

Summary of reduction function $f$:
Given formula, make $G$ with T-F-N triangle, 1 pair of literal nodes per variable, 2 “or” gadgets per clause, connected as in example.
Note: again, $f$ does not know or construct satisfying assignment or coloring.
Correctness:
• Show $f$ poly time computable: A key point is that graph size is polynomial in formula size; graph looks messy, but pattern is basically straightforward.
• Show $c$ in 3-SAT iff $f(c)$ is 3-colorable:
  $(\Rightarrow)$ Given an assignment satisfying $c$, color literals T/F as per assignment; can color “or” gadgets so output nodes are T since each clause is satisfied.
  $(\Leftarrow)$ Given a 3-coloring of $f(c)$, name colors T-N-F as in example. All square nodes are T or F (since all adjacent to N). Each variable pair $(x_i, \neg x_i)$ must have complementary labels since they’re adjacent. Define assignment based on colors of $x_i$’s. Clause “output” nodes must be colored T since they’re adjacent to both N & F. By fact noted earlier, output can be T only if at least one input is T, hence it is a satisfying assignment.
Coping with NP-hardness
Coping with NP-Hardness

Is your real problem a special subcase?
   E.g. 3-SAT is NP-complete, but 2-SAT is not; ditto 3- vs 2-coloring
   E.g. only need planar-/interval-/degree 3 graphs, trees,…?

Guaranteed approximation good enough?
   E.g. Euclidean TSP within $1.5 \times \text{Opt}$ in poly time

Fast enough in practice (esp. if $n$ is small),
   E.g. clever exhaustive search like dynamic programming, backtrack, branch & bound, pruning

Heuristics – usually a good approx and/or fast
NP-complete problem: TSP

Input: An undirected graph $G=(V,E)$ with integer edge weights, and an integer $b$.

Output: YES iff there is a simple cycle in $G$ passing through all vertices (once), with total cost $\leq b$. 

Example: 

$b = 34$
TSP - Nearest Neighbor Heuristic

Recall NN Heuristic—go to nearest unvisited vertex

Fact: NN tour can be about \((\log n) \times \text{opt}\), i.e.

\[
\lim_{n \to \infty} \frac{NN}{OPT} \to \infty
\]

(above example is not that bad)
2x Approximation to Euclidean TSP

A TSP tour visits all vertices, so contains a spanning tree, so cost of min spanning tree < TSP cost.

Find MST

Find “DFS” Tour

Shortcut

TSP ≤ shortcut < DFST = 2 * MST < 2 * TSP
1.5x Approximation to Euclidean TSP

Find MST (solid edges)

Connect odd-degree tree vertices (dotted)

Find min cost matching among them (thick)

Find Euler Tour (thin)

Shortcut (dashed)

Shortcut \( \leq ET \leq MST + TSP/2 < 1.5 \ast TSP \)

\[ \text{Cost of matching} \leq TSP/2 \]

(next slide)
Matching $\leq$ TSP/2

Oval = TSP

Big dots = odd tree nodes
(Exercise: show every graph has an even number of odd degree vertices)

Blue, Green = 2 matchings

Blue + Green $\leq$ TSP (triangle inequality)

So min matching $\leq$ TSP/2
Progress on TSP approximation

This 1.5x approximation was the best known for $\approx 35$ years

New CSE faculty member Shayan Gharan with collaborators Saberi and Singh improved on this recently; you might enjoy watching the recording of the colloquium he gave on this in April, 2013:

**New Approximation Algorithms for the Traveling Salesman Problem**

(http://www.cs.washington.edu/events/colloquia/search/details?id=2360)
P / NP Summary
Many important problems are in $\mathbf{P}$: solvable in deterministic polynomial time

Details are the fodder of algorithms courses. We’ve seen a few examples here, plus many other examples in other courses

Few problems not in $\mathbf{P}$ are routinely solved;

For those that are, practice is usually restricted to small instances, or we’re forced to settle for approximate, suboptimal, or heuristic “solutions”

A major goal of complexity theory is to delineate the boundaries of what we can feasibly solve
NP

The tip-of-the-iceberg in terms of problems conjectured not to be in P, but a very important tip, because

a) they’re very commonly encountered, probably because
b) they arise naturally from basic “search” and “optimization” questions.

Definition: poly time verifiable;
“guess and check”, “is there a…” – are also useful views
NP-completeness

Defn & Properties of $\leq_p$

A is NP-hard: everything in NP reducible to A
A is NP-complete: NP-hard and in NP

“the hardest problems in NP”
“All alike under the skin”

Most known natural problems in NP are complete

#1: 3CNF-SAT

Many others: Clique, IndpSet, 3Color, KNAP, HamPath, …
Summary

Big-O – good
P – good
Exp – bad
Exp, but hints help? NP
NP-hard, NP-complete – bad (I bet)
To show NP-complete – reductions
NP-complete = hopeless? – no, but you need to lower your expectations: heuristics, approximations and/or small instances.
Common Errors in NP-completeness Proofs

Backwards reductions

   Bipartiteness $\leq_p$ SAT is true, but not so useful.
   (XYZ $\leq_p$ SAT shows XYZ in NP, doesn’t show it’s hard.)

Sloooow Reductions

   “Find a satisfying assignment, then output…”

Half Reductions

   E.g., after removing one of the “slack” weights in the
   KNAP reduction, still true that KNAP sol $\Rightarrow$ SAT sol, but
   no longer vice versa. Adding another slack does opposite.
“I can’t find an efficient algorithm, but neither can all these famous people.”

[Garay & Johnson, 1979]
Thus, for any nondeterministic Turing machine M that runs in some polynomial time \( p(n) \), we can devise an algorithm that takes an input \( \omega \) of length \( n \) and produces \( E_{M,\omega} \). The running time is \( O(p^2(n)) \) on a multitape deterministic Turing machine and...

WTF, MAN. I JUST WANTED TO LEARN HOW TO PROGRAM VIDEO GAMES.