Many problems of our interest are search problems with exponentially (or even infinitely) many solutions

- Shortest of the paths between two vertices
- Spanning tree with minimal cost
- Combination of variable values that minimize an objective

We should be surprised we find efficient (i.e., polynomial-time) solutions to these problems

- It seems like these should be the exceptions rather than the norm!

What do we do when we hit upon other search problems?
Hard Problems: Where you find yourself ...

I can’t find an efficient algorithm, I guess I’m just too dumb.

Images from “Computers and Intractability” by Garey and Johnson
Search and Optimization Problems

- What do we do when we hit upon hard search problems?
- Can we prove they can’t be solved efficiently?
I can’t find an efficient algorithm, because no such algorithm is possible.

Images from “Computers and Intractability” by Garey and Johnson
Search and Optimization Problems

- Unfortunately, it is very hard to prove that efficient algorithms are impossible.
- Second best alternative:
  - Show that the problem is as hard as many other problems that have been worked on by a host of brilliant scientists over a very long time.
- Much of complexity theory is concerned with categorizing hard problems into such *equivalence classes*. 

\textit{P, NP, Co-NP, NP-hard and NP-complete}
Nondeterminism and Search Problems

- Nondeterminism is an oft-used abstraction in language theory
  - Non-deterministic FSA
  - Non-deterministic PDA

- So, why not non-deterministic Turing machines?
  - Acceptance criteria is analogous to NFA and NPDA
    - if there is a sequence of transitions to an accepting state, an NDTM will take that path.

- What does nondeterminism, a theoretical construct, mean in practice?
  - You can think of it as a boundless potential to search for and identify the correct path that leads to a solution
  - So, it does not change the class of problems that can be solved, just the time/space needed to solve.
Class \textit{NP}: Non-deterministic Polynomial Time

How they operate:

- Guess a solution
- verify correctness in polynomial time

Polynomial time verifiability is the key property of \textit{NP}.

- This is how you build a path from \textit{P} to \textit{NP}.
- Ideal formulation for search problems, where correct solutions are hard to find but easy to recognize.

Example: Boolean formula satisfiability (\textit{SAT})

- Given a boolean formula in CNF, find an assignment of \{\text{true}, \text{false}\} to variables that makes it true.
- Why not DNF?
What are the bounds of $NP$?

- **Only Decision problems:**
  - Problems with an “yes” or “no” answer
  - Optimization problems are generally not in $NP$
    - But we can often find optimal solutions using “binary search”

- **“No” answers are usually not verifiable in $P$-time**
  - So, complement of $NP$ problems are often not $NP$.
  - $UNSAT$ — show that a CNF formula is false for all truth assignments\(^1\)

- **Key point:** You cannot negate nondeterministic automata.
  - So, we are unable to convert an NDTM for $SAT$ to solve $UNSAT$ in $NP$-time.

\(^1\)Whether $UNSAT \in NP$ is unknown!
What are the bounds of \( NP \)?

- **Existentially quantified vs Universally quantified formulas**
  - \( NP \) is good for \( \exists \bar{x} \ P(\bar{x}) \): guess a value for \( \bar{x} \) and check if \( P(\bar{x}) \) holds.
  - \( NP \) is not good for \( \forall \bar{x} \ P(\bar{x}) \): guessing does not seem to help if you need to check all values of \( \bar{x} \).

- Negation of existential formula yields a universal formula.
  - No surprise that complement of \( NP \) problems are typically not in \( NP \).
  - \( UNSAT \): \( \forall \bar{x} \neg P(\bar{x}) \) where \( P \) is in CNF
  - \( VALID \): \( \forall \bar{x} P(\bar{x}) \), where \( P \) is in DNF

- \( NP \) seems to be a good way to separate hard problems from even harder ones!
Co-NP: Problems whose complement is in NP

- Decision problems that have a polynomially checkable proof when the answer is “no”

What we think the world looks like.

- Biggest open problem: Is \( P = NP \)?
- Will also imply \( co-NP = P \)
The class $\text{Co-NP} \cap \text{NP}$

- Often, problems that are in $\text{NP} \cap \text{co-NP}$ are in $\text{P}$
- It requires considerable insight and/or structure in the problem to show that something is both $\text{NP}$ and $\text{co-NP}$
- This can often be turned into a $\text{P}$-time algorithm

Examples
- Linear programming [1979]
  - Obviously in $\text{NP}$. To see why it is in $\text{co-NP}$, we can derive a lower bound by multiplying the constraints by a suitable (guessed) number and adding.
- Primality testing [2002]
  - Obviously in $\text{co-NP}$; See “primality certificate” for proof it is $\text{NP}$
- Integer factorization?
A problem \( \Pi \) is \( NP \)-hard if the availability of a polynomial solution to \( \Pi \) will allow \( NP \)-problems to be solved in polynomial time.

\( \Pi \) is \( NP \)-hard \( \iff \) if \( \Pi \) can be solved in \( P \)-time, \( P = NP \)

\( NP \)-complete = \( NP \)-hard \( \cap \) \( NP \)

More of what we think the world looks like.
Polynomial-time Reducibility

- Show that a problem $A$ could be transformed into problem $B$ in polynomial time
  - Called a polynomial-time reduction of $A$ to $B$
  - The crux of proofs involving $NP$-completeness

**Implication:** if $B$ can be solved in $P$-time, we can solve $A$ in $P$-time

- An $NP$-complete problem is one to which any problem in $NP$ can be reduced to.

**Never forget the direction:** To prove a problem $\Pi$ is $NP$-complete, need to show how all other $NP$ problems can be solved using $\Pi$, not vice-versa!
Wait! How can I reduce every NP to my problem?

- If a particular NP-problem A is given to you, then you can think of a way to reduce it to your problem B.
- But how do you go about proving that every NP problem X can be reduced to B?
  - You don’t even know X — indeed, the class NP is infinite!

If you already knew an NP-complete problem, your task is easy!

- Simply reduce this NP-complete problem to B, and by transitivity, you have a reduction of every $X \in \text{NP}$ to B.

So, who will bell the cat?

- Stephen Cook [1970] and Leonid Levin [1973] managed to do this!
- Cook was denied reappointment/tenure in 1970 at Berkeley, but won the Turing award in 1982!
The first $NP$-complete problem: $SAT$

How do you show reducibility of arbitrary $NP$-problems to $SAT$? You start from the definition, of course!

- The class $NP$ is defined in terms of an NDTM
  - $X$ is in $NP$ if there is an NDTM $T_X$ that solves $X$ in polynomial time

- Use this NDTM as the basis of proof.

Specifically, show that acceptance by an NDTM can be encoded in terms of a boolean formula

- Model $T_X$ tape contents, tape heads, and finite state at each step as a vector of boolean variables
  - Need $(p(n))^2$ variables, where $p(n)$ is the (polynomial) runtime of $T_X$

- Model each transition as a boolean formula
Thanks to Cook-Levin, you can say ...

I can’t find an efficient algorithm, but neither can all these famous people.

Thanks to $NP$-completeness results, you can say this even if you have been working on an obscure problem that no one ever looked at!
Some Hard Decision Problems
Traveling Salesman Problem

Given \( n \) vertices and \( n(n - 1)/2 \) distances between them, is there a \textit{tour} (i.e., cycle) of length \( b \) or less that passes through all vertices?
Hamiltonian Cycle

- Simpler than TSP
  - Is there a cycle that passes through every vertex in the graph?
- Earliest reference, posed in the context of chess boards and knights (“Rudrata cycle”)
- Longest path is another version of the same problem
  - When posed as a decision problem, becomes the same as Hamiltonian path problem
Balanced Cuts

Does there exist a way to partition vertices $V$ in a graph into two sets $S$ and $T$ such that

1. there are at most $b$ edges between $S$ and $T$, and
2. $|S| \geq |T| \geq |V|/3$
Integer Linear Programming (ILP) and Zero-One Equations (ZOE)

**ILP:** Linear programming, but solutions are limited to integers
- Many problems are easy to solve over real numbers but much harder for integers.
- Examples:
  - Knapsack
  - Solutions to equations such as $x^n + y^n = z^n$

**ZOE:** A special case of ILP, where the values are just 0 or 1.
- Find $x$ such that $Ax = 1$ where $1$ is a column matrix consisting of 1’s.
3d-Matching

- Given triples of compatibilities between men, women and pets, find perfect, 3-way matches.
Independent set, vertex cover, and clique

Independent set: Does this graph contain a set of at least $k$ vertices with no edge between them?

Vertex cover: Does this graph contain a set of at least $k$ vertices that cover all edges?

Clique: Does this graph contain at least $k$ vertices that are fully connected among themselves?
# Easy Vs Hard Problems

<table>
<thead>
<tr>
<th>Hard</th>
<th>Easy</th>
</tr>
</thead>
<tbody>
<tr>
<td>3SAT</td>
<td>2SAT, HORN SAT</td>
</tr>
<tr>
<td>TSP</td>
<td>MST</td>
</tr>
<tr>
<td>Longest path</td>
<td>Shortest path</td>
</tr>
<tr>
<td>3d-matching</td>
<td>bipartite match</td>
</tr>
<tr>
<td>Independent set</td>
<td>Indep. set on trees</td>
</tr>
<tr>
<td>ILP</td>
<td>Linear programming</td>
</tr>
<tr>
<td>Hamiltonian cycle</td>
<td>Euler path, Knights tour</td>
</tr>
<tr>
<td>Balanced cut</td>
<td>Min-cut</td>
</tr>
</tbody>
</table>
**NP-completeness: Polynomial-time Reductions**

- Show that a known \( NP \)-complete problem \( A \) could be transformed into problem \( B \) in polynomial time

\[ \text{Instance } I \xrightarrow{f} \text{Instance } f(I) \xrightarrow{\text{Algorithm for } A} \text{Solution } S \text{ of } f(I) \xrightarrow{h} \text{Solution } h(S) \text{ of } I \]

- **Implication:** if \( B \) can be solved in \( P \)-time, we can solve \( A \) in \( P \)-time

- **Never forget the direction:**
  - We are proving that \( B \) is \( NP \)-complete here.
NP-completeness Reductions

All of NP

- SAT

- 3SAT

- Independent set

- Vertex cover

- Clique

- 3D matching

- Subset sum

- ILP

- Rudrata cycle

- ZOE

- TSP
Reducing all of \( NP \) to \( SAT \)

- We already discussed this
  - Show how to reduce acceptance by an NDTM to the \( SAT \) problem.

- \textit{Exercise:} Show how to transform acceptance by an FSA into an instance of \( SAT \)
Reducing $SAT$ to $3SAT$

- $3SAT$: A special case of $SAT$ where each clause has $\leq 3$ literals
- Reduction involves transforming a disjunction with many literals into a CNF of disjunctions with $\leq 3$ literals per term
- The transformation below at most doubles the problem size.

**Key Idea:** Introduce additional variables:

- Example: $l_1 \lor l_2 \lor l_3 \lor l_4$ can be transformed into:
  
  $$(l_1 \lor l_2 \lor y_1) \land (\overline{y_1} \lor l_3 \lor l_4)$$

  For this conjunction to be true, one of $\{l_1, \ldots, l_4\}$ must be true:
  
  So a solution to the transformed problem is a solution to the original — simply discard assignments for the new variables $y_i$. 
Reducing 3SAT to Independent set

- Nontrivial reduction, as the problems are quite different in nature
- **Idea:** Model each of \( k \) clauses of 3SAT by a “triangle” in a graph

The graph corresponding to \((x \lor y \lor z) (x \lor \overline{y} \lor z) (x \lor y \lor z) (x \lor \overline{y})\).

- Independent set of size \( k \) must contain one literal from each clause
  - By setting that literal to *true*, we obtain a solution for 3SAT
- **Key point:** Avoid conflicts, e.g., assigning *true* to both \( x \) and \( \overline{x} \)
  - ensure using edges between every variable and its complement
Reducing Independent set to Vertex Cover

If $S$ is an independent set then $V - S$ is a vertex cover

- Consider any edge $e$ in the graph
- **Case 1:** Both ends of $e$ are in $V - S$
- **Case 2:** At least one end of $e$ is $S$. The other end of $e$ cannot be in $S$ or else $S$ won’t be independent.

Thus, in both cases, at least one side of $e$ must go to $V - S$.

In other words $V - S$ is a vertex cover

Thus, we have reduced independent set to vertex cover problem.
Reducing Independent set to Clique

- If $S$ is an independent set then $S$ is clique in $\overline{G} = (V, \overline{E})$
  - For any pair $v_1, v_2 \in S$ there is no edge in $E$
    - means that there is an edge between any such pair in $G'$
    - i.e, $S$ is a clique in $\overline{G}$

- Thus, we have reduced independent set to the clique problem, while only using polynomial time and space.
NP-completeness Reductions

- We have discussed the left half of this picture.
- We won’t discuss the right half, since the proofs are similar in many ways, but are more involved.
- You can find those reductions in the text book.

![Reductions diagram]

- All of NP
  - SAT
  - 3SAT
  - 3D MATCHING
  - ZOE
  - RUDRATA CYCLE
  - TSP
  - ILP
  - SUBSET SUM
  - VERTEX COVER
  - CLIQUE
  - INDEPENDENT SET
Beyond NP: PSPACE

- **PSPACE**: The class of problems that can be solved using only polynomial amount of space.
  - It is OK to take exponential (or super-exponential) time.

- **Key point**: Unlike time, space is reusable.
  - Result: many exponential algorithms are in PSPACE.
    - Consider universal formulas. We can check them in polynomial space by rerunning the same computation (say, \( \text{check}(v) \)) for each \( v \).
    - The space used for \( \text{check} \) is recycled, but the time adds up for different \( v \)'s.

- **Note**: \( SAT \) is in PSPACE
  - Try every possible truth assignment for variables.

- Thus, all \( NP \)-complete problems are in PSPACE.
PSPACE-hard and PSPACE-complete

**PSPACE-hard:** A problem \( \Pi \) is PSPACE-hard if for any problem \( \Pi' \) in PSPACE there is a \( P \)-time reduction to \( \Pi \).

**PSPACE-complete:** PSPACE-hard problems that are in PSPACE.

- **Examples:**
  - **QBF:** Quantified boolean formulae
  - **NFA totality:** Does this NFA accept all strings?

**Is \( NP \subseteq \text{PSPACE} \)?**

- We think so, but we can’t even prove \( P \subsetneq \text{PSPACE} \)
The class EXP (aka EXPTIME) consists of the class of problems that can be solved in $O(2^{n^k})$ time for some $k$.

PSPACE $\subseteq$ EXP.

Intuitively, you can’t do more than EXP work using a PSPACE algorithm because you need polynomial amount of space even if the only thing you did is to count up to $2^n$.

As usual, EXP-hard and EXP-complete are defined using $P$-time reductions.

Generalized versions of games such as chess and checkers are EXP-hard.

We think PSPACE $\subsetneq$ EXP, but can only prove $P \subsetneq EXP$. 
Where do we stop?

- These classes can be extended for ever:
  - **NEXP**: Nondeterministic exponential time
  - **EXPSPACE**: Problems solvable with exponential space.
  - **EEXP**: Problems solvable in double exp. time \(O(2^{2^{(n^k)}})\) for some \(k\)

- **Examples**:
  - Equivalence of regexpr with intersection is EXPSPACE-hard.
  - REs with negation can’t be decided even in \(E^k\)EXPTIME for any \(k\).

- **Inclusions**:
  - \(P \subseteq NP \subseteq PSPACE \subseteq EXP \subseteq NEXP \subseteq EXPSPACE \subseteq EEXP \subseteq NEEXP \subseteq \text{EEXPSPACE} \subseteq \cdots\)

- We *think* these classes are distinct, but have proofs only for classes that are 3 places apart, e.g., \(P\) and \(EXP\).