7: Complexity Theory

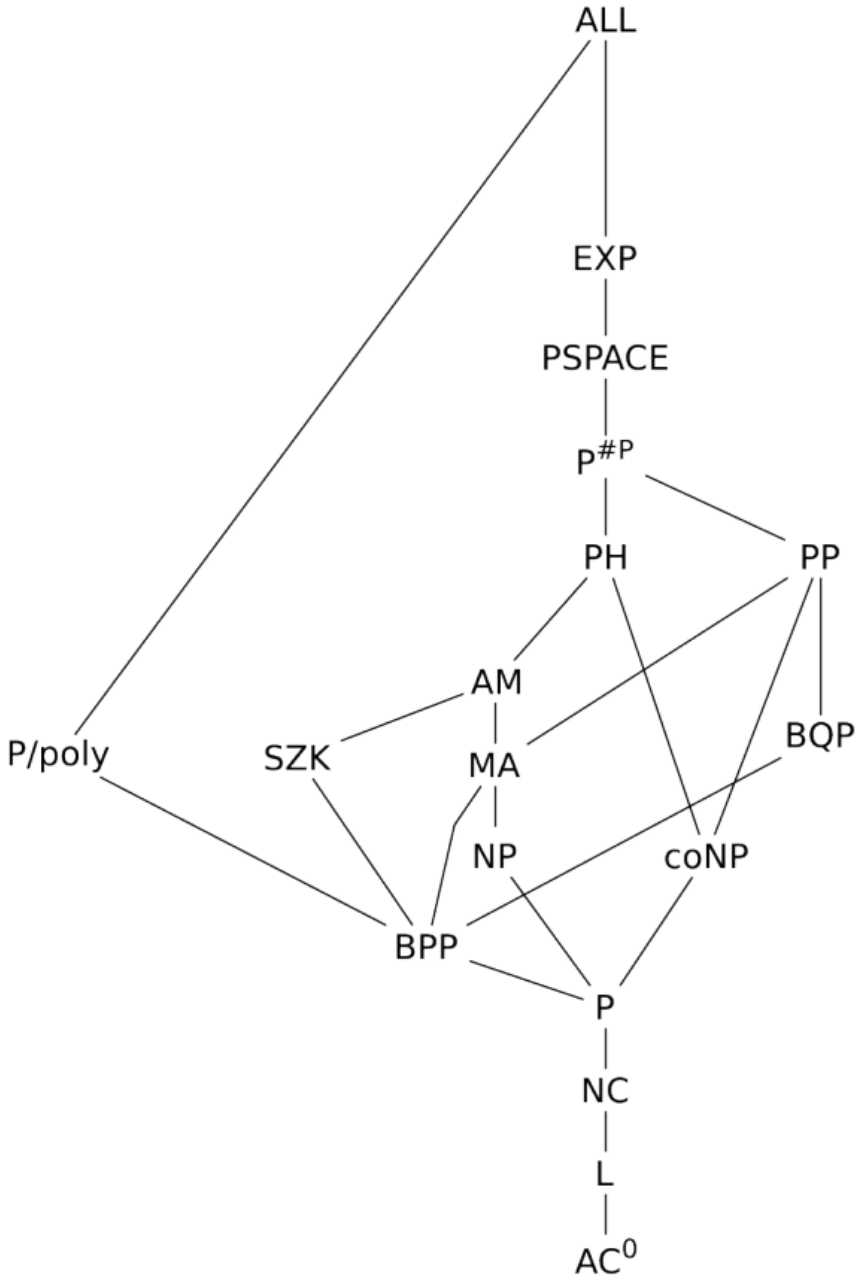
CS500 Design and Analysis of Algorithms

Algorithms vs Problems

- We talked about algorithms
  - Gale–Shapley, Prim, Kruskal, Schwartz–Zippel, …
- Now we talk about problems
  - Stable matching, Minimum spanning tree, Polynomial identity testing, …

Complexity class

- "Set of problems with related resource-based complexity"
- Examples:
  P, **NP**, **PSPACE**, **EXPTIME**, …
- Complexity Zoo



Complexity class **P**

- **P** = Set of all decision problems that are decidable with a polynomial-time algorithm
- What is "decision problem"?
- What is "decidable"?
- What is "polynomial-time"?

Problem

- A relation from a set of inputs to a set of outputs
- Example: STABLE-MATCHING
  - **Input:** Lists of preferences for k men and k women
  - **Output:** A stable matching between the men and the women
- Example: SUBSET-SUM
  - **Input:** A list of integers, and an integer k
  - **Output:** A nonempty subset that sums to k

Decision problem

- A function from a set of inputs to {YES, NO}
- Example: EXISTS-STABLE-MATCHING
  - **Input:** Lists of preferences for k men and k women
  - **Output:** Does there exist a stable matching?
- Example: EXISTS-SUBSET-SUM
  - **Input:** A list of integers, and an integer k
  - **Output:** Does there exist a nonempty subset that sums to k?
- STABLE-MATCHING and SUBSET-SUM are not decision problems
  - Function problems: set of outputs might be larger than just 2 elements

Encoding

- Set of inputs: Matrices (stable marriage), Graphs (minimum spanning tree), Lists of integers (subset sum, sorting), …
- Can be encoded as a binary string
  - E.g. Write in ASCII
- Can be encoded as a natural number (0, 1, 2, 3, …)
  - E.g. Append a 1 in the front, treat as binary representation
- Length of input x: $\ell(x) = \lfloor \log_2 x \rfloor + 1$ and $\ell(0) = 0$
  - Write x in binary
  - Not in unary

Decision problem

- A function from a set of inputs to {YES, NO}
- A function from the set of natural numbers to {YES, NO}
  - Because we can encode inputs to natural numbers
- A subset of natural numbers
  - Take the set of inputs that map to YES

Complexity class **P**

- **P** = Set of all decision problems that are decidable with a polynomial-time algorithm
- ~~What is "decision problem"?~~
- What is "decidable"?
- What is "polynomial-time"?

Decidability

- Has an algorithm that "solves" it
  - With problem $L \subseteq I \times O$: on input $x \in I$, terminates and outputs any $y \in O$ s.t. $(x, y) \in L$
- For decision problem L: on input x, terminates and outputs YES if $x \in L$, otherwise NO
- We only talk about decidable problems
  - For more about computation theory, see CS422: Theory of Computation

Polynomial-time algorithm

- Polynomial-time algorithm: exists polynomial p s.t. on input x, terminates in at most $p\big(\ell(x)\big)$ steps
- Gale–Shapley: $O(k^2)$ where k is number of men/women
  - $O(n)$ where n is length of input
- Trial division for primality testing: $O(x)$
  - Not polynomial! $x \approx 2^{\ell(x)}$ so it's $O(2^n)$ where n is length of input
  - AKS algorithm: $O(n^7)$
- Decidable in polynomial-time: has an algorithm that both decides it and runs in polynomial-time

Complexity class **P**

- **P** = Set of all decision problems that are decidable with a polynomial-time algorithm
  - EXISTS-STABLE-MATCHING in **P**?
    - Yes because Gale–Shapley returns a stable matching in polynomial-time
    - Yes because we can always answer "yes"
  - Gale–Shapley in **P**?
    - No because **P** is set of decision problems, not algorithms
  - STABLE-MATCHING in **P**?
    - No because STABLE-MATCHING is a function problem, not decision problem
- Tractable: we can solve them "quickly"

Complexity classes

- **P** = Set of all decision problems that are decidable with a polynomial-time algorithm
- **PSPACE** = Set of all decision problems that are decidable with a polynomial-space algorithm
- **EXPTIME** = Set of all decision problems that are decidable with a exponential-time algorithm

PSPACE? EXPTIME?

- Polynomial-time algorithm: exists polynomial p s.t. on input n, terminates in at most $p\big(\ell(n)\big)$ steps
- Exponential-time algorithm: exists polynomial p s.t. on input n, terminates in at most $2^{p\left(\ell(n)\right)}$ steps
- Polynomial-space algorithm: exists polynomial p s.t. on input n, terminates and uses at most $p\big(\ell(n)\big)$ bits of memory

$\mathbf{P} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXPTIME}$

- $\mathbf{P} \subseteq \mathbf{PSPACE}$: Every step can use at most one additional bit of memory
- $\mathbf{PSPACE} \subseteq \mathbf{EXPTIME}$: With m bits of memory, runs for at most 2^m steps; if a configuration is repeated then program doesn't halt
- $\mathbf{P} = \mathbf{PSPACE}$? $\mathbf{PSPACE} = \mathbf{EXPTIME}$?
  - We don't know!
  - Yes: More time/space resources don't help
  - No: Some problems are inherently difficult (need lots of resources)

Complexity classes

- **P** = Set of all decision problems that are decidable with a polynomial-time algorithm
- **NP** = Set of all decision problems that are verifiable with a polynomial-time algorithm
- **PSPACE** = Set of all decision problems that are decidable with a polynomial-space algorithm
- **EXPTIME** = Set of all decision problems that are decidable with a exponential-time algorithm

Verifiable problems

- Consider decision problem L
- Verifiable: there exists decidable L' s.t.
  - On inputs $x \in L$, has a witness w where $(x, w) \in L'$
  - On inputs $x \notin L$, doesn't have such witness
- Verifier of L: algorithm that decides L'
- Analogy
  - Proofs in mathematics (L = "true theorem", verifier = proof checker)
  - Property of people (L = "exists a student here satisfying this property", witness = said person)

Complexity class **NP**

- **NP** = Set of all decision problems that are verifiable with a polynomial-time algorithm
- Witness must be polynomial in length
  - Exists polynomial p s.t. on input $x \in L$, witness w satisfies $\ell(w) \leq p\big(\ell(x)\big)$
- $L \in$ **NP**: there exists $L' \in$ **P** s.t. on input $x \in L$, exists poly-length witness w where $(x, w) \in L'$, and otherwise no such witness

EXISTS-SUBSET-SUM

- **Input:** A list of integers, and an integer k
- **Output:** Does there exist a nonempty subset that sums to k?
- Witness: subset that add to k
  - Verifier: check if the numbers come from the input and they add up to k
  - Verifier might say NO, but that's because of bad witness
  - Witness is poly-length?
    - Yes, it's at most as long as input
  - Verifier runs in poly-time?
    - Yes, it's linear time
- EXISTS-SUBSET-SUM is in **NP**

$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE}$

- $\mathbf{P} \subseteq \mathbf{NP}$: Verifier doesn't read witness, decides input right away
- $\mathbf{NP} \subseteq \mathbf{PSPACE}$: Witness is poly-length, so try every possible witness
- $\mathbf{P} = \mathbf{NP}$?
  - We don't know!
  - If you prove or disprove it, you get $1,000,000
  - Yes: everything that can be verified quickly can be decided quickly
  - No: some problems are easy to check but hard to solve
  - Mathematicians and theoretical computer scientists think $\mathbf{P} \neq \mathbf{NP}$

Example: Travelling Salesman

- Hamiltonian cycle: a cycle that visits all vertices
- **Input:** Weighted graph G, integer k
- **Output:** Does there exist a Hamiltonian cycle of total weight $\leq k$?
- Witness: Hamiltonian cycle of total weight $\leq k$
  - Verifier: check it visits all vertices, check total weight $\leq k$
  - Witness of poly-length? Verifier runs in poly-time?
    - Check them yourself

More examples: Eulerian circuit and Hamiltonian cycle

- Eulerian circuit: a closed walk that visits all edges exactly once
- EULERIAN-CIRCUIT: Does there exist an Eulerian circuit in G?
- HAMILTONIAN-CYCLE: Does there exist a Hamiltonian cycle in G?
- Both are in **NP** (witness: the asked thing)
- But EULERIAN-CIRCUIT is in **P**
  - Theorem: A graph has an Eulerian circuit iff it is connected excluding isolated vertices and every vertex has even degree
    - Not proven here
  - Hierholzer's algorithm finds one in linear-time

More examples: Evaluating formula and satisfiability

- Boolean formula: I hope you know
  - E.g. 0, $x \vee \neg x$, $(x \wedge y) \rightarrow (x \leftrightarrow y)$, …
- EVAL: Given Boolean formula φ and assignments to variables, is the result true?
- SAT: Given Boolean formula φ, does there exist an assignment to variables that makes the result true?
- Both are in **NP**, but EVAL is also in **P**

Even more examples

One of these is proven in **P**;
the rest are in **NP**
but we don't know if in **P**

- EC (Edge Cover): In graph G, do there exist $\leq k$ edges s.t. every vertex is incident to at least one selected edge?
- VC (Vertex Cover): In graph G, do there exist $\leq k$ vertices s.t. every edge is incident to at least one selected vertex?
- CLIQUE: Does graph G have a k-clique?
- IS (Independent Set): Does graph G have an independent set of size k?
- ILP (Integer Linear Programming): Given $A \in \mathbb{Z}^{n \times m}, B \in \mathbb{Z}^{m}$, does there exist $x \in \mathbb{N}^{n}$ s.t. $Ax = B$?

Comparing decision problems

- Which one is "harder"?
  - IS-SQUARE: Given (a, b), is $b = a^2$?
  - IS-PRODUCT: Given (a, b, c), is $c = ab$?
- IS-PRODUCT is harder
  - $(a, b) \in$ IS-SQUARE implies $(a, a, b) \in$ IS-PRODUCT
  - If we can solve IS-PRODUCT, we can also solve IS-SQUARE

Many-one reduction $L \leq L'$

- Exists $f : \mathbb{N} \to \mathbb{N}$ s.t. $x \in L \Leftrightarrow f(x) \in L'$
  - When $L =$ IS-SQUARE and $L' =$ IS-PRODUCT, take $f(a, b) = (a, a, b)$
- L is reducible to L'
- f is called the reduction
- If $L \leq L'$ and $L' \leq L''$ then $L \leq L''$
  - Compose the two reductions

Polynomial-time reduction $L \leq_p L'$

- Exists $f : \mathbb{N} \to \mathbb{N}$ s.t. $x \in L \Leftrightarrow f(x) \in L'$
- The reduction f can be computed in poly-time
- We will only talk about polynomial-time reductions
- If $L' \in \mathbf{P}$ and $L \leq_p L'$ then $L \in \mathbf{P}$
  - Given x and algorithm A that decides L' in poly-time:
    1. Compute $f(x)$
    2. Call A on input $f(x)$
    3. Return result
  - We have algorithm that decides L in poly-time

CLIQUE vs IS

- CLIQUE: Given graph G, does it have k-clique?
- IS: Given graph G, does it have independent set of size k?
- CLIQUE $\leq_p$ IS: $f(G,k) = \left(\overline{G},k\right)$ (graph complement)
  - If $(G,k) \in$ CLIQUE then $\left(\overline{G},k\right) \in$ IS
  - If $\left(\overline{G},k\right) \in$ IS then $(G,k) \in$ CLIQUE
  - f can be computed in poly-time
- IS $\leq_p$ CLIQUE: $f(G,k) = \left(\overline{G},k\right)$
- CLIQUE and IS are "equally hard"

NP-completeness

- **NP**-hard problem L': any problem $L \in \mathbf{NP}$ is polynomial-time reducible to L'
- If <u>any</u> **NP**-hard problem is in **P**, then <u>all</u> problems in **NP** are in **P**
  - $L \leq_p L'$, since $L' \in \mathbf{P}$ then $L \in \mathbf{P}$, but $L \in \mathbf{NP}$
- Means $\mathbf{NP} \subseteq \mathbf{P}$ and hence $\mathbf{P} = \mathbf{NP}$
- We believe $\mathbf{P} \neq \mathbf{NP}$
- If a problem is **NP**-hard, it's probably not in **P**
  - Means it's hard
- **NP**-complete: in **NP** and also **NP**-hard

NP-completeness

- **NP**-hard problem L': any problem $L \in \mathbf{NP}$ is polynomial-time reducible to L'
- If a problem is **NP**-hard, it's probably not in **P**
- If $L' \leq_p L''$ and L' is **NP**-hard, then L'' is **NP**-hard
  - Take $L \in \mathbf{NP}$, then $L \leq_p L' \leq_p L''$
- If we have an **NP**-hard problem and we reduce it to another problem, the latter is also **NP**-hard
  - Easy way to show a problem is probably hard
  - Instead of proving <u>every</u> **NP** problem is reducible to L'', just need <u>one</u>

The original (**NP**-)hard problem

- 3-CNF Boolean formula: conjunction of disjunctions of 3 literals (variable or negation of it)
  - E.g. $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_4 \vee \neg x_5) \wedge (\neg x_2 \vee \neg x_3 \vee \neg x_4)$
  - Each disjunction of 3 literals is a clause
- 3-SAT: Given 3-CNF formula φ, does there exist an assignment of variables so it's true?
  - Any assignment needs at least one literal in each clause to be true
  - 3-SAT $\in$ **NP**: exercise
  - Cook–Levin theorem: 3-SAT is **NP**-hard (and hence **NP**-complete)
    - Direct proof, not by reduction; not covered here

3-SAT $\leq_p$ IS

- $\varphi = (t_{11} \vee t_{12} \vee t_{13}) \wedge (t_{21} \vee t_{22} \vee t_{23}) \wedge \cdots \wedge (t_{k1} \vee t_{k2} \vee t_{k3})$
- Ideas:
  - A vertex for each term
  - Independent set shows one term that is true in each clause
- Graph G has $3k$ vertices: (i, j) for $1 \leq i \leq k$ and $j = 1, 2, 3$
- Distinct vertices (i_1, j_1) and (i_2, j_2) are adjacent if and only if:
  - $i_1 = i_2$: terms in the same clause can't be chosen together
  - $t_{i_1 j_1} = \neg t_{i_2 j_2}$: a variable and its negation can't be chosen together
- Reduction: $f(\varphi) = (G, k)$

3-SAT $\leq_p$ IS

- f is a poly-time reduction
  - $\varphi \in$ 3-SAT means $f(\varphi) \in$ IS
    - From a satisfying assignment, take one true literal from each clause
  - $f(\varphi) \in$ IS means $\varphi \in$ 3-SAT
    - Assign variables so everything in the independent set is true (and the remaining variables whatever)
  - f is poly-time
    - Obvious
- Since 3-SAT is **NP**-hard, IS is **NP**-hard
- CLIQUE is also **NP**-hard

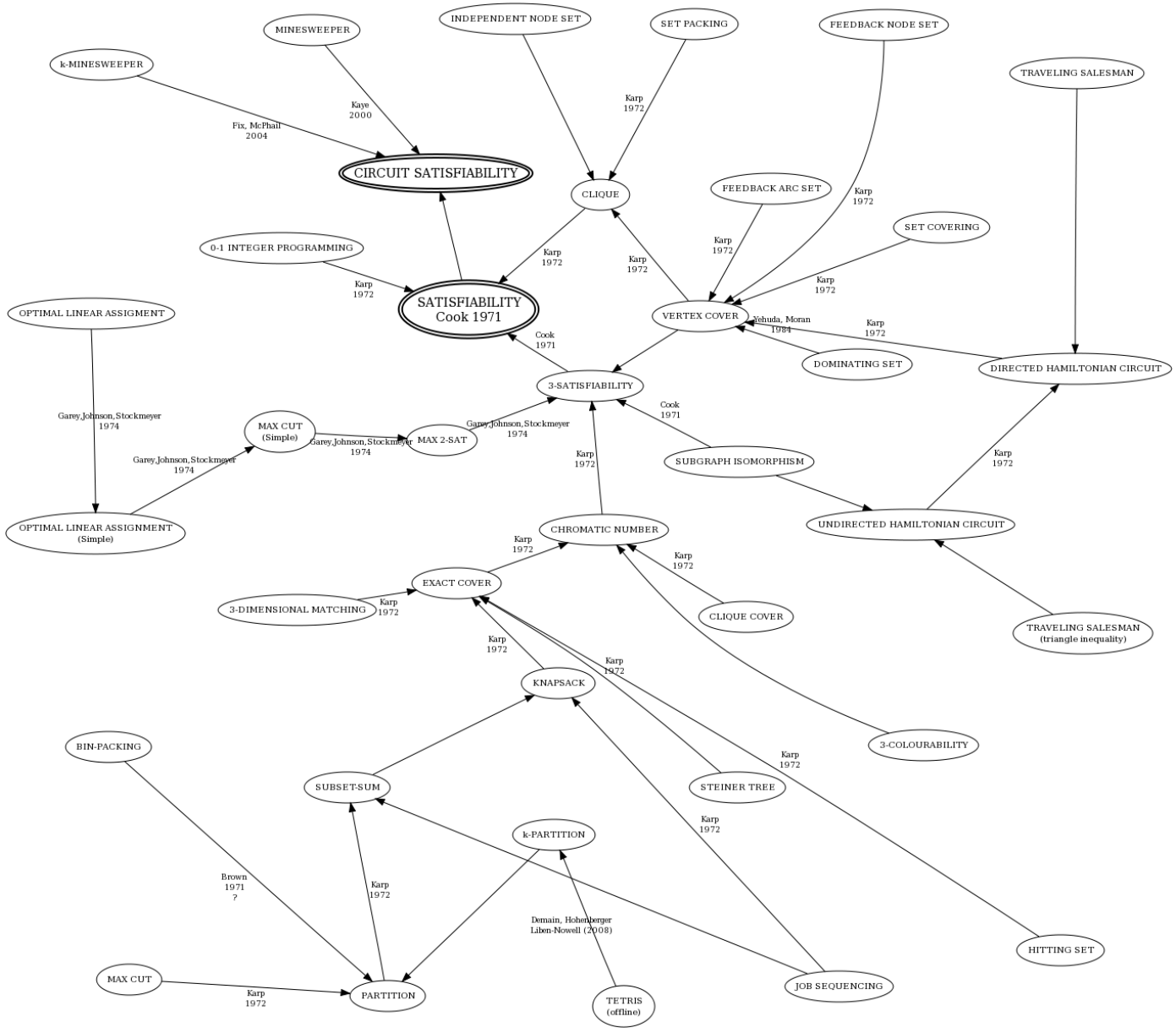
IS $\leq_p$ VC

- IS: Given graph G, does it have independent set of size k?
- VC: Given graph G, do there exist $\leq k$ vertices so every edge is incident to at least one selected vertex?
- $f(G, k) = (G, |V| - k)$
  - Take the complement of an independent set to get a vertex cover
  - Proof: exercise
- VC is also **NP**-hard

Many **NP**-complete problems

- 3-SAT
- CLIQUE
- INDEPENDENT-SET
- VERTEX-COVER
- EXISTS-SUBSET-SUM
- TRAVELLING-SALESMAN
- HAMILTONIAN-CYCLE
- INTEGER-LINEAR-PROGRAMMING
- 3-COLORING
- KNAPSACK
- SUDOKU
- …

[Wikipedia]



How to solve hard problems

- Exact solution?
  - Too long (recall **NP**-complete problems are probably not in **P**)
- Randomization
  - Often works, sometimes fails
- Approximation
  - Not the exact solution, just a "close enough" solution
  - Next time: VC is **NP**-hard but we can get an approximate solution
    - If the optimal vertex cover has size k, we can find a vertex cover of size at most $2k$ in poly-time