NP Part 2

Language

- A **language L** over Σ is a subset of Σ^*.
- Σ is the alphabet
- Example $\Sigma=\{0,1\}$
- $L=\{10, 11, 101, 111, 1011, 1101, 10001, \ldots\}$
  – What is L?
  – What is $\overline{L}=\Sigma^*-L$?
- L_1 concatenated with L_2
  – $L=\{ x_1x_2 : x_1 \in L_1 \text{ and } x_2 \in L_2\}$

Language

- $L^*=\{\lambda\} \cup L \cup L^2 \cup L^3 \cup \ldots$
  - Called the closure or Kleene star
- An algorithm A accepts $x \in \{0,1\}^*$, if given input x, the output is 1, i.e. $A(x)=1$.
- An algorithm A rejects $x \in \{0,1\}^*$, if given input x, the output is 0, i.e. $A(x)=0$.
- The language accepted by algorithm A is the set of strings $L= \{x \in \{0,1\}^* : A(x)=1\}$

Language

- The language accepted by algorithm A is the set of strings L= $\{x \in \{0,1\}^* : A(x)=1\}$
- For $x \notin L$, A will not necessarily reject x, i.e. we are not guaranteed that $A(x)=0$, but that $A(x) \neq 1$
  - Example A my loop forever given input x
- A language L is **<u>accepted in polynomial</u> time** by A if it is accepted by A and if $|x|=n$, there exists a constant k so that $T(n)<=O(n^k)$
- A language L is **<u>decided in polynomial time</u>** by A given $x \in \{0,1\}^*$, A determines if $x \in L$ or $x \notin L$ and if $|x|=n$, there exists a constant k so that $T(n)<=O(n^k)$

Complexity Class

- Each Complexity class is a set of languages.
- P={L $\subseteq$ {0,1}* : there exist an algorithm A that decides L in polynomial time}
- Theorem 34.2
  - P={L $\subseteq$ {0,1}* : there exist an algorithm A that **accepts** L in polynomial time}
  - A accepts x $\in$ L in at most cn^k steps, for some c, k and |x|=n
  - After cn^k , if A has not accepted x, then A(x)=0

Verification Algorithms

- Given G=(V,E) a hamiltonian cycle is a simple cycle which contains every vertex of V.
- Formal Language
  - Ham_Cycle= { <G>: G is a hamiltonian graph}
- Verification, given G, suppose someone said G is hamiltonian and they offer proof.
  - a certificate (ordered list of vertices)
  - We can verify this is $O(n^2)$ time
  - Input is G and y (the certificate)

Verification Algorithms

- Verification algorithm
  - A inputs x and a certificate y, A(x,y).
  - If y verifies x, then A(x,y)=1
- A language verified by a verification algorithm

$L=\{x \in \{0,1\}^* : \text{there exists } y \in \{0,1\}^* \text{ for which } A(x,y)=1\}$

- So for any $x \in L$, there is a y for which A(x,y)=1
- For any $x \notin L$ and for any y, $A(x,y) \neq 1$

NP (nondeterministic polynomial)

- The complexity class NP is the set of languages which can be verified by a polynomial time algorithm.
- $L \in NP \Leftrightarrow$

  $L=\{x \in \{0,1\}^* :$ there exists y with $|y|=O(x^c)$ (c a constant) such that $A(x,y)=1\}$
- A **verifies** language L in **polynomial time**

Complexity

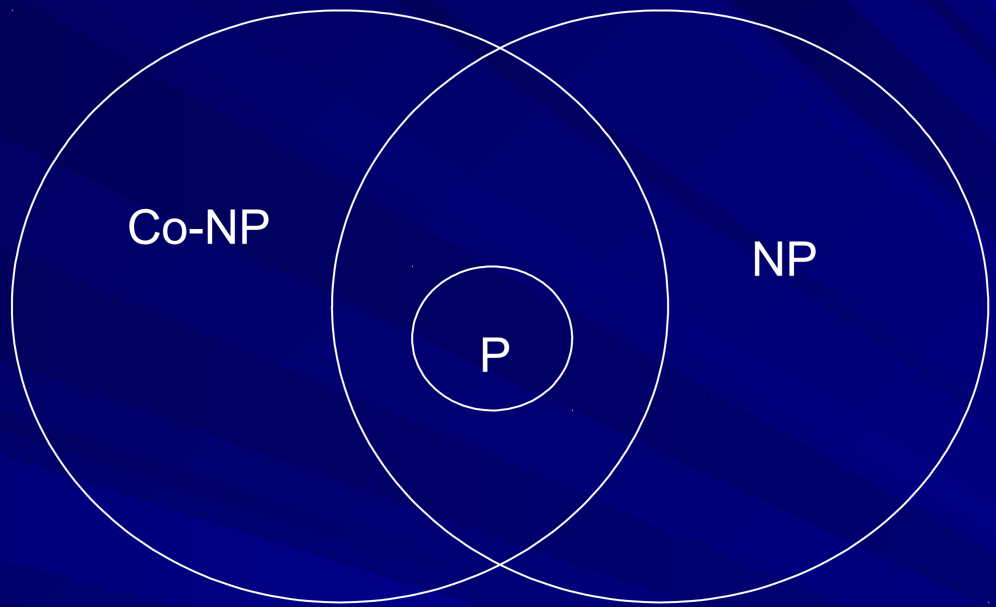
- $P \subseteq NP$
  - Given $L \in P$, there is a algorithm A which accepts L in polynomial time.
  - Covert algorithm A(x) to algorithm A(x,y) and A ignores y.
  - A(x,y)=1 if A(x)=1, for any y
  - A(x,y)=0 if A(x)=0, for any y

Complexity

- P = NP?
  - Unknown (probably not).
  - Clay, $1,000,000 dollar prize for solution
- One of the largest if not largest unsolved problem in Computer Science

Co-NP

- co-NP={ $\overline{L}$, for $L \in$ NP}
- Example L={primes}
  - What is $\overline{L}$?
- Unknown co-NP=NP?
  - i.e. is NP
- P is closed under complement
  - P $\subseteq$ co-NP
  - P $\subseteq$ co-NP$\cap$NP
- Unknown P=co-NP$\cap$NP?



- 4 possible cases
  - P=NP=co-NP
  - NP=co-NP,   P≠NP
  - P=NP∩P, NP ≠ co-NP
  - P ≠ NP∩P, NP ≠ co-NP

Tractable and Intractable

- Outside of NP, print the numbers from 1 to n^n
- A problem is tractable if there is a polynomial time algorithm which solves it.
- A problem is intractable if it is either undecidable or all algorithms are of exponential time.
- An undecidable example is the Halting problem

Halting Problem

- Given an algorithm A and input string x, determine if A will stop.
- Halt(A,x)= true if A stops
- Halt(A,x)= false if A enters infinite loop
- Can there exist a algorithm Halt?
- trouble(*string* x)
  - **if** halt(A, x) = **false return true**
  - **else** loop forever

Halting Problem

- trouble(*string* x)
  - **if** halt(A, x) = **false return true**
  - **else** loop forever
- A=trouble, x=trouble
- Suppose Halt (trouble, trouble) is false
  - So Trouble(trouble) did not halt
  - But then if Halt (trouble, trouble) was false, the if statement is satisfied and trouble returns true
    - So halt(trouble, trouble) is true (trouble stopped)
    - contradiction

Reducibility

- L_1 is polynomial-time reducible to L_2, written $L_1 \leq_P L_2$ (or $L_1 \propto L_2$) if there exists a polynomial-time computable function f:{0,1}*---> {0,1}* such that for all $x \in \{0,1\}^*$,

  $x \in L_1 \Leftrightarrow f(x) \in L_2$

- If $L_1 \propto L_2$, we can use L_2 to solve L_1.
  - We create an algorithm A_1 which solves L_1
  - Let A_2 be an algorithm which solves L_2
  - For $x \in \{0,1\}^*$, compute f(x) (polynomial time)
  - Output A_2(f(x)) [this is A_1(x)]
  - Running time A_1 is polynomial if A_2 is polynomial.

NP-Complete

- $L \in$ NP-complete
  - $L \in NP$
  - for all $L' \in NP$, $L' \propto L$
- NP-complete is the hardest problems in NP. Solving any NP-complete problem efficiently would give a solution to any problem in NP.

- NP-hard
  - for all $L' \in NP$, $L' \propto L$
  - L may not necessarily be in NP

Show the following

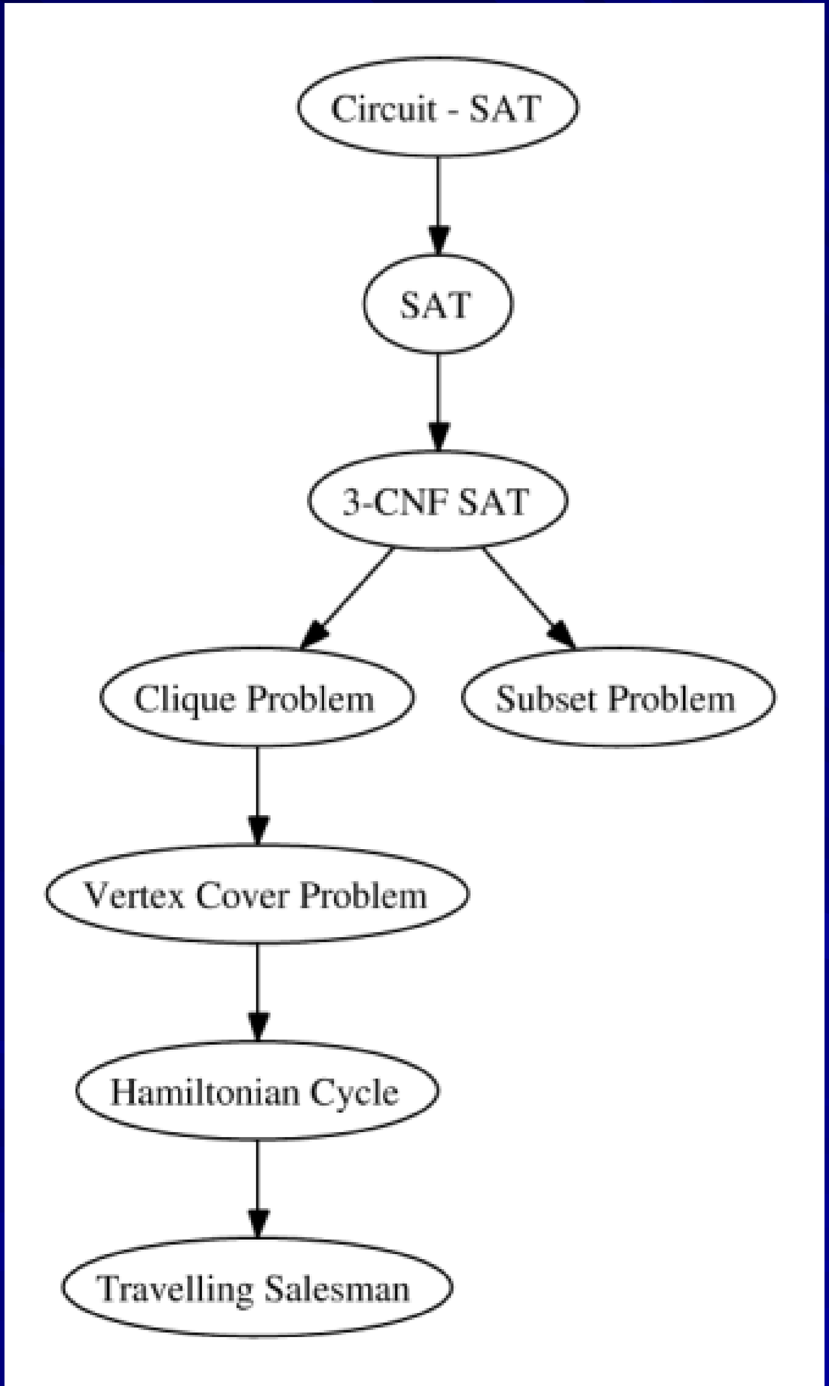
- If $Q \propto L$ and $L \in P$, then $Q \in P$
- If $L_1, L_2 \in NP$, $L_1 \in NP$-complete and $L_1 \propto L_2$, then $L_2 \in NP$-complete.
- If $L \in NP$-complete and $L \in P$, then $P=NP$.
- If $L \in NP$ and $L \notin P$, then $P \neq NP$.

Showing L is NP-complete

- Show $L \in NP$.
- Find $Q \in$ NP-complete and show $Q \propto L$.
- This assumes we know a problem in NP-Complete.
- Finding a first NP-complete problem is hard.

Cook’s Theorem

- 1971 Satisfiability Problem is NP-Complete
- V set of Boolean variables.
- C collection of clauses, well-formed formulas.
- Problem- “Is there a satisfying truth assignment for C?”


Circuit - SAT
SAT
3-CNF SAT
Clique Problem
Subset Problem
Vertex Cover Problem
Hamiltonian Cycle
Travelling Salesman

Informal NP-Complete Proof

- Longest Path is a NP-Complete Problem
  - Solve the Hamiltonian Cycle Problem
  - Formulate as NP problem, (yes no problem)
  - Given a graph G, vertices u and v, and a number k, does there exist a simple **path** from s to t with at least k edges?
  - Input G=(V,E), u, v, k.
  - Given a certificate, we can check in polynomial time if it is a path of length k from s to t
- For each edge (u,v),
  - Call LP(V,E-(u,v), u, v, n-1)
- At most n^2 edges so we call LP at most n^2 times.