CS 121 Section 10:
Poly-time Reductions, NP, and the Cook Levin Thm.

Fall 2020

Prof: Madhu Sudan and Adam Hesterberg
TF: Eric Lin

Context for today

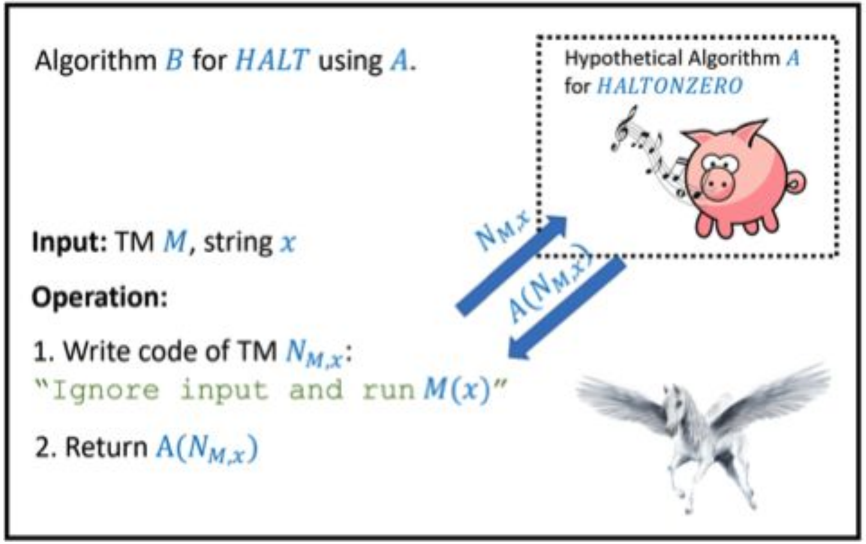
- Where did we come from?
  - P vs. EXP
  - Uncomputability: reductions and Rice’s Theorem
- Where are we now?
  - Revisiting reductions with the added notion of time complexity
    - Chapter 14: “Polynomial time reductions” (link to text)
  - NP, NP completeness, and Cook Levin Theorem
    - Chapter 15 (link to text)
- Where are we going?
  - Explore more classes of Functions and algorithms
    - Randomized algorithms

Polynomial-time Reductions

Past: Reductions to prove Uncomputability

In order to prove that *HALTONZERO* is uncomputable, we show *HALT* ≤ *HALTONZERO* :

1. Compute *HALT* using *HALTONZERO*
   a. *HALT*(M,x) = *HALTONZERO*(M)
   b. *HALT(M,x) = HALTONZERO(G(M,x))* where *G* is a reduction function to transform the inputs of *HALTONZERO* to *HALT*
2. Show correctness
   a. Soundness and completeness



Ex: Reducing *HALT* to *HALTONZERO*.

Polynomial-time Reductions

In order to show $A \leq B$:

1. Compute *A* using *B*
   a. *A*(M,x) = *B*(M)
   b. *A(M,x) = B(G(M,x))* where *G* is a reduction function to transform the inputs of *B* to *A*
2. Show correctness
   a. Soundness and completeness

In order to show $A \leq_p B$:

1. Compute *A* using *B*
   a. *A*(M,x) = *B*(M)
   b. *A(M,x) = B(R(M,x))* where *R* is a polynomial-time computable reduction function that transforms the inputs of *B* to *A*
2. Show correctness
   a. Soundness and completeness
   b. Show *R* can be computed in polynomial-time

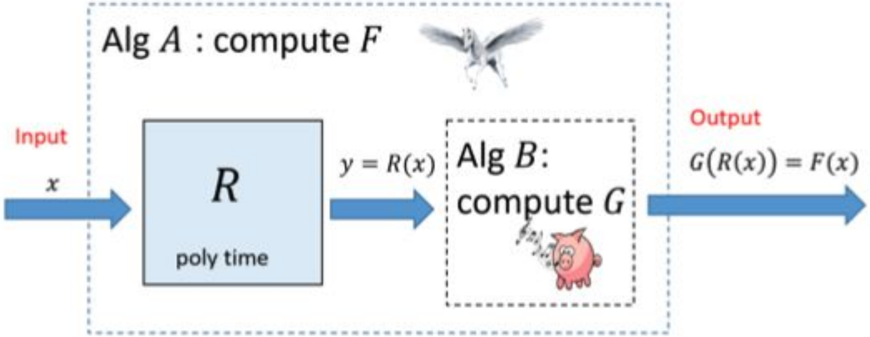
Formal Definition

Definition 14.1 (Polynomial-time reductions)

Let $F, G : \{0,1\}^* \rightarrow \{0,1\}$. We say that F *reduces to* G, denoted by $F \leq_p G$ if there is a polynomial-time computable $R : \{0,1\}^* \rightarrow \{0,1\}^*$ such that for every $x \in \{0,1\}^*$,

$$F(x) = G(R(x)) \ . \quad (14.2)$$

We say that F and G have *equivalent complexity* if $F \leq_p G$ and $G \leq_p F$.



Why do we care?

Proving $A \leq_p B$ leads to this stipulation:

If B can be computed with a poly-time algorithm,
Then A is also computable in poly-time.

(Concept check: why can't we say this with the regular reduction $A \leq B$?)

NP

Class of NP Functions

Definition 15.1 (NP)

We say that $F : \{0,1\}^* \to \{0,1\}$ is in $\mathbf{NP}$ if there exists some integer $a > 0$ and $V_F : \{0,1\}^* \to \{0,1\}$ such that $V_F \in \mathbf{P}$ and for every $x \in \{0,1\}^n$,

$$F(x) = 1 \Leftrightarrow \exists_{w \in \{0,1\}^{n^a}} \text{ s.t. } V_F(xw) = 1 \,. \quad (15.1)$$

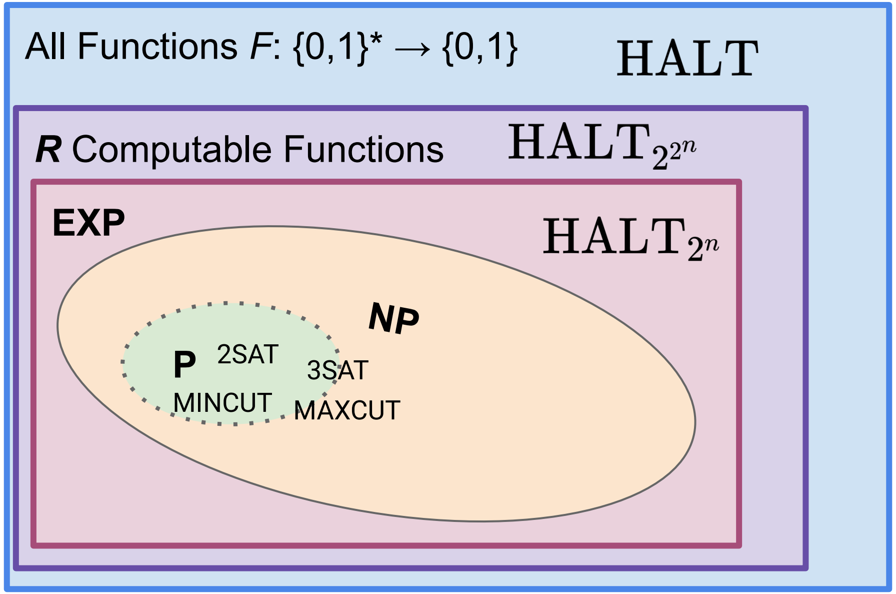
Non-mathematical explanation:

- **NP** is the class of functions that are efficiently verifiable.

NP := Non-deterministic Polynomial-time

NP ≠ Not Polynomial-time

Concept Check: Prove NP $\subseteq$ EXP



NP Hardness and Completeness

NP-hard

- A function *F* is NP-hard if all functions in NP can be reduced to it
  - *F* is "at least as hard" as all functions in NP; $\forall G \in \mathrm{NP},\ G \leq_p F$

NP-complete

- A function *F* is NP-complete if
  1. *F* is in NP
  2. *F* is NP-hard

Cook–Levin Theorem

Cook–Levin Theorem

Once we prove $A \leq_p B$, we know that:

- If *B* is in NP then *A* is also in NP.
- If *A* is NP-hard then *B* is also NP-hard.

Theorem 15.6 (Cook-Levin Theorem)

For every $F \in \mathbf{NP}$, $F \leq_p 3SAT$.

=> *3SAT* is NP-complete

Key Concept: Given *F* and *x*, we use V_F and *x* to get a small circuit *C* s.t. $C(w) = 1$ iff $V_F(x,w) = 1$.

Example Reductions

Zero-One Linear Equations Problem

Problem: show that there is no efficient polynomial-time algorithm to compute the *Zero-One Linear Equations Problem*.

The *Zero-One Linear Equations Problem* corresponds to the function $01EQ : \{0,1\}^* \rightarrow \{0,1\}$ where the input is a collection $= E$ of linear equations in variables $x_0, ...x_{n-1}$, and the output is 1 iff $\exists$ assignment $x \in \{0,1\}^n$ of 0/1 values to the variables that satisfies all the equations. So if E encodes the equations $x_0 + x_1 + x_2 = 2, x_0 + x_2 = 1, x_1 + x_2 = 2$ then $01EQ = 1$ because there exists an assignment to satisfy this ($x = 011$).

Ideas?

Zero-One Linear Equations Problem

Problem: show that there is no efficient polynomial-time algorithm to compute the *Zero-One Linear Equations Problem*.

To prove this, we can reduce a known NP-Hard problem (i.e. 3SAT) to Zero-One Linear Equations to show 3SAT $\leq_p$ 01EQ.

Steps for a Poly-time Reduction Proof

1. Describe a reduction function R to transform the inputs
2. Show R can be computed in polynomial time
3. Show Correctness
   a. Completeness
   b. Soundness

Step 1: Describe a reduction function R

We want to convert the inputs of 3SAT to the inputs of 01EQ

$(x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_5 \vee x_6)$ -> system of 01 linear equations

We need to constrain each variable.

Hint: make each clause an equation. Any ideas on approaching this?

Step 1: Describe a reduction algorithm R

We want to convert the inputs of 3SAT to the inputs of 01EQ

$(x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_5 \vee x_6)$ -> system of 01 linear equations

Proof idea: A constraint $x_2 \vee \bar{x_5} \vee x_7$ can be rewritten as $x_2 + (1 - x_5) + x_7 \geq 1$.
Making it an equation: Since the sum of the left hand side is an inequality but can be at most 3, we add *auxiliary variables* to make it an equality.
Dealing with negated variables: We also add another variable x'_i to correspond to the negation of x_i and include the equation $x_i + x'_i = 1$.
Thus we have transformed

$$x_2 \vee \bar{x_5} \vee x_7 \implies x_2 + x'_5 + x_7 + y + z = 3$$

More generically, we transform:

$$x_1 \vee x_2 \vee x_3 \implies x_1 + x_2 + x_3 + u + v = 3$$

Step 2: Show R runs in polynomial time

- Initial loop of n steps to set up constraint for each variable
  - Each iteration takes constant time
- Another loop of m steps
  - Each iteration also taking constant time to convert a clause into an equation

Step 3a: Show Completeness

Step 3a: Show completeness: If $3SAT(\varphi) = 1$ **then** $01EQ(R(\varphi) = 1$

This is the first part of our proof of correctness. Suppose that $3SAT(\varphi) = 1$, then there is an assignment w to satisfy φ. Every clause in φ has form $w_1 \vee w_2 \vee w_3$ so because $w_1 + w_2 + w_3 \geq 1$ we can represent it as $w_1 + w_2 + w_3 + y + z = 3$ where y and z are between 0 and 1. If we let $x'_i = 1 - x_i$ for each variable i, the assignment $x_0 \cdots x_{n-1}, x'_0 \cdots x'_{n-1}, y_0 \cdots y_{m-1}, z_0 \cdots z_{m-1}$ satisfies $E = R(\varphi)$ so $01EQ(R(\varphi)) = 1$.

Step 3b: Show soundness

Step 3b: Show soundness: If $01EQ(R(\varphi) = 1$ then $3SAT(\varphi) = 1$

This is the second part of our proof of correctness. Suppose that $01EQ(R(\varphi) = 1$. Then there must be some assignment $x_0 \cdots x_{n-1}, x'_0 \cdots x'_{n-1}, y_0 \cdots y_{m-1}, z_0 \cdots z_{m-1}$.

Based on the way we did our transformation R, we know that x'_i is the negation of x_i for all $i \in [n]$. Because we defined $y_j, z_j \in [0, 1]$, $y_j + z_j \leq 2$ for all j in $[m]$. Thus, for every clause C_j in φ of the form $w_1 \vee w_2 \vee w_3$, we have $w_1 + w_2 + w_3 \geq 1$. This means the assignment $x_0 \cdots x_{n-1}$ satisfies φ and thus $3SAT(\varphi) = 1$.

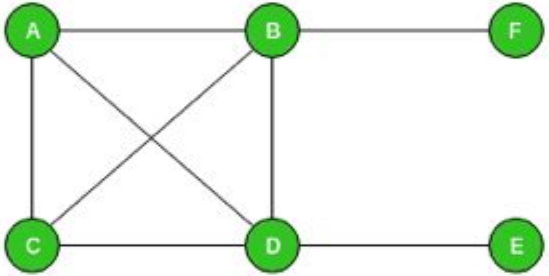
Another Example

Clique

Definition: Given an undirected graph $G = (V, E)$, a clique is a subset $V' \subseteq V$ *s.t.* $(v_1, v_2) \in E$ for all $v_1, v_2 \in V'$.

Consider the function *CLIQUE(G, k)* = 1 iff G has a clique of size k, and 0 otherwise.

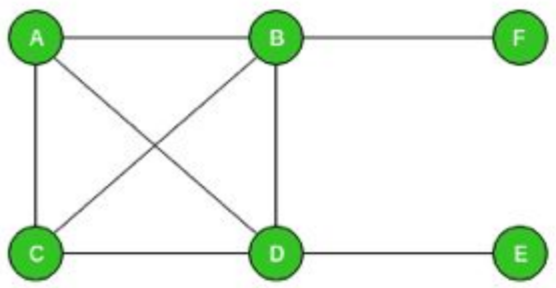
Is *CLIQUE* **NP**-complete? Prove it or show why not.



Clique

Reminder: To prove NP-completeness, we need to:

1. Show *CLIQUE* $\in$ NP



2. Show *CLIQUE* is NP-hard by reducing another NP-hard problem to *CLIQUE* (e.g. 3SAT $\leq_p$ *CLIQUE*).

Clique

Reminder: To prove NP-completeness, we need to:

1. Show *CLIQUE* $\in$ NP.
2. Show *CLIQUE* is NP-hard by reducing another NP-hard problem to *CLIQUE* (e.g. 3SAT $\leq_p$ *CLIQUE*).

Some NP-hard problems:

1. 3SAT
2. Min. k-cut
3. ISET
4. HALT

Steps for a Poly-time Reduction Proof

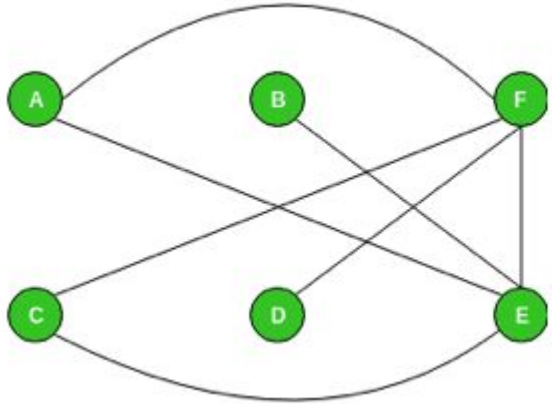
1. Describe a reduction function R to transform the inputs
2. Show R can be computed in polynomial time
3. Show Correctness
   a. Completeness
   b. Soundness

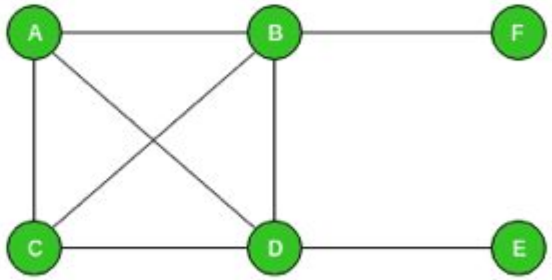
Step 1: Describe a reduction function R

We want to convert the inputs of *ISET* to the inputs of *CLIQUE*

ISET(G(V,E), k)=1 iff G contains an indep. set of size ≥ k.

CLIQUE(G’(V’,E’), k)=1 iff G’ contains a clique of size ≥ k.



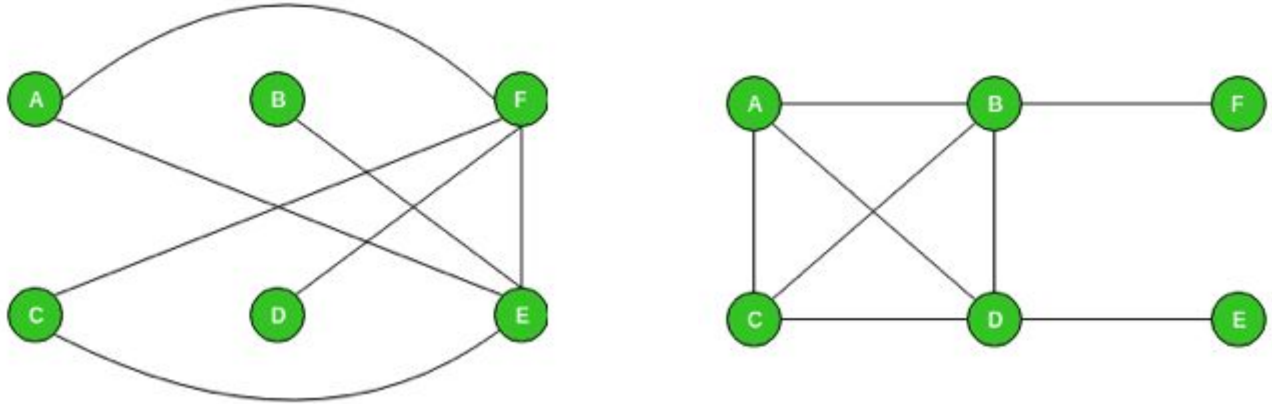


Step 1: Describe a reduction function R

We want to convert the inputs of *ISET* to the inputs of *CLIQUE*

ISET(G(V,E), k)= CLIQUE(R(G,k))

where R(G,k) returns the complement of G and keeps k the same

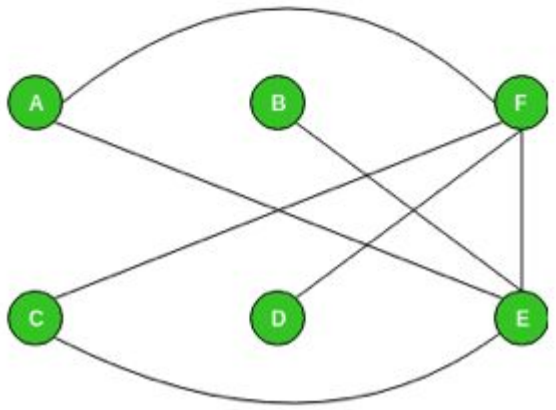


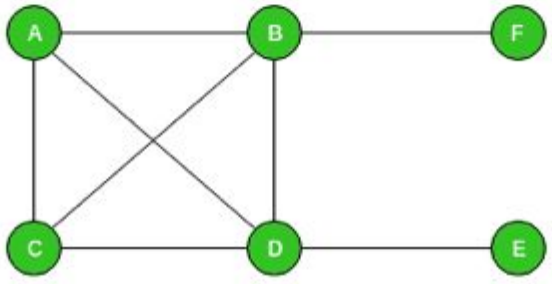
Step 2: Show R runs in polynomial time

- Converting G to its complement G’ takes O(|V| + |E|) time.

Step 3a: Show Completeness

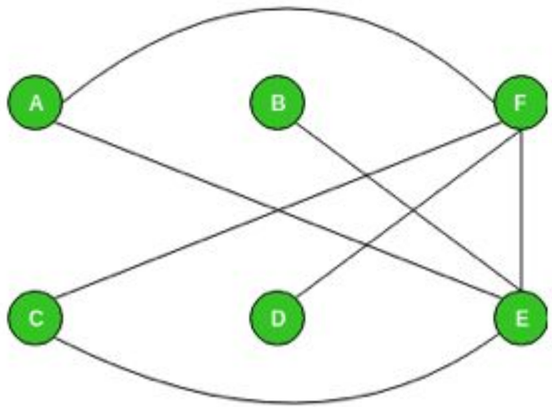
Show: if *ISET*(G, k) = 1 then *CLIQUE(G'*, k) = 1.

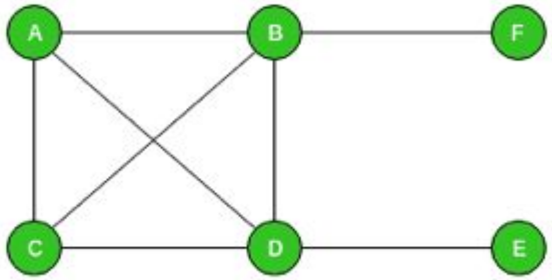




Step 3b: Show soundness

Show: if *CLIQUE*(G', k) = 1 then *ISET*(G, k) = 1.

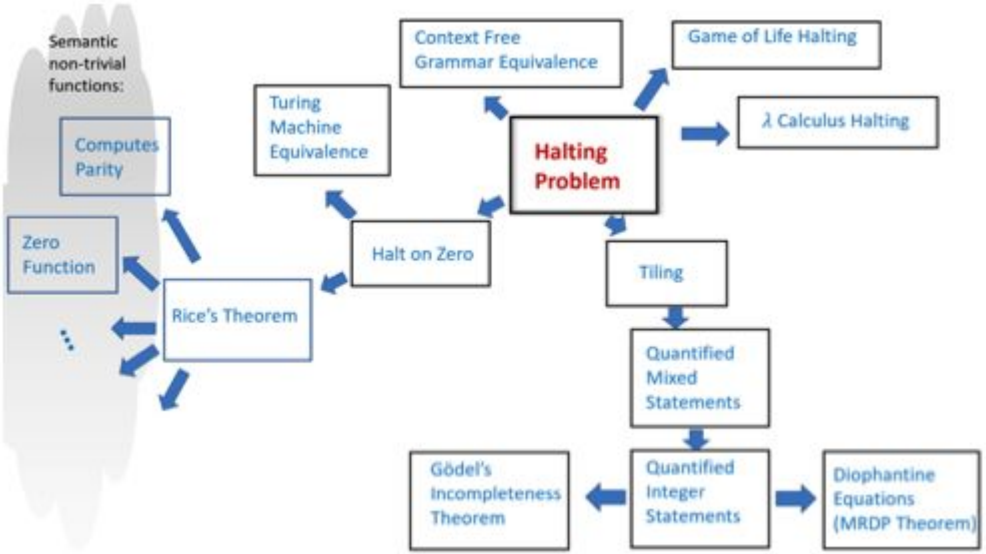


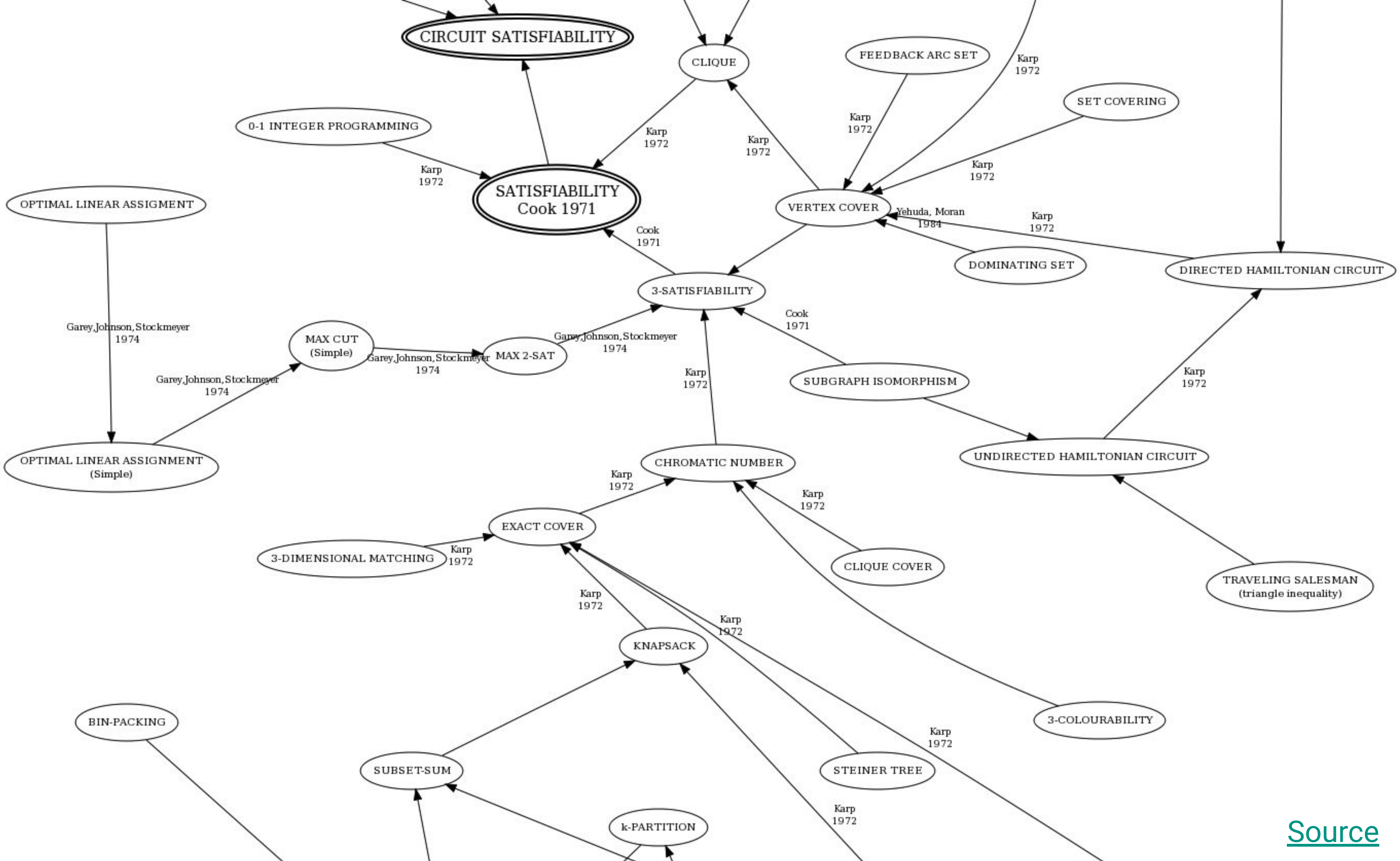


Implications of All of This

Reminder: Uncomputability

Once proven $A \leq B$: If A is uncomputable then B is also uncomputable.




CIRCUIT SATISFIABILITY
CLIQUE
FEEDBACK ARC SET
Karp
1972
SET COVERING
0-1 INTEGER PROGRAMMING
Karp
1972
Karp
1972
Karp
1972
Karp
1972
Karp
1972
SATISFIABILITY
Cook 1971
OPTIMAL LINEAR ASSIGMENT
VERTEX COVER
Yehuda, Moran
1984
Karp
1972
Cook
1971
DOMINATING SET
DIRECTED HAMILTONIAN CIRCUIT
3-SATISFIABILITY
Cook
1971
Garey,Johnson,Stockmeyer
1974
MAX CUT
(Simple)
Garey,Johnson,Stockmeyer
1974
MAX 2-SAT
Garey,Johnson,Stockmeyer
1974
Karp
1972
SUBGRAPH ISOMORPHISM
Karp
1972
Garey,Johnson,Stockmeyer
1974
OPTIMAL LINEAR ASSIGNMENT
(Simple)
CHROMATIC NUMBER
UNDIRECTED HAMILTONIAN CIRCUIT
Karp
1972
Karp
1972
EXACT COVER
3-DIMENSIONAL MATCHING
Karp
1972
CLIQUE COVER
TRAVELING SALESMAN
(triangle inequality)
Karp
1972
Karp
1972
KNAPSACK
BIN-PACKING
3-COLOURABILITY
Karp
1972
SUBSET-SUM
STEINER TREE
k-PARTITION
Karp
1972

What if P = NP?

- Difference between proving P = NP and actually discovering / constructing a “fast” algorithm
- Public-key cryptography
  - Passwords are verifiable in poly-time
- Automating discovery of mathematical proofs
- Videos to check out
  - Richard Karp on difficulty of proving P=NP
  - Donald Knuth’s intuition on P=NP
  - Scott Aaronson on what happens if P=NP

More Problems

1. Transitivity of Poly-time Reductions

Show that for every $F, G, H : \{0,1\}^* \to \{0,1\}$, if $F \leq_p G$ and $G \leq_p H$, then $F \leq_p H$.

2. Vertex–Cover

Given an undirected graph $G = (V, E)$, a vertex-cover is a subset $V' \subseteq V$ *s.t.* for all $(v_1, v_2) \in E$, either $v_1 \in V'$ or $v_2 \in V'$. Consider the function *VERTEX-COVER(G, k)* = 1 iff G has a vertex cover of size k, and 0 otherwise. Is *VERTEX-COVER* **NP**-complete? Prove it or show why not.

3. Set-Cover NP-Completeness

Given n sets $S_1, S_2, \ldots, S_n$ such that

$$\bigcup_{i=1}^{n} S_i = A$$

the set cover of size k over these sets is a collection C of k of these sets such that

$$\bigcup_{i \in C} S_i = A$$

Given a collection of sets and an integer k, SET-COVER returns if there exists a valid set cover of a most size k over the given collection of sets. Prove that SET-COVER is NP-complete.

4. Say if P, NP, or Uncomputable:

(a) Given an integer x, determine if x has a prime factor that is at most k.

(b) Given an undirected graph graph, determine whether it is possible to partition its vertices into two sets, with at least k edges crossing between sets.

(c) Given a program Q, an input x, and a string 1^t, determine whether Q halts on x within t steps.

Thank You!

Appendix

coNP

Show that coNP = NP

Define $F \in \mathsf{coNP}$ iff $\overline{F} \in \mathsf{NP}$, where $\overline{F}$ denotes the negation of the output of F (for example, if $F(00) = 1$, then $\overline{F}(00) = 0$). Prove that if $\mathsf{P} = \mathsf{NP}$, then $\mathsf{coNP} = \mathsf{NP}$.