CS 121: Lecture 19
NP and NP-Completeness

Madhu Sudan

https://madhu.seas.Harvard.edu/courses/Fall2020

Book: https://introtcs.org

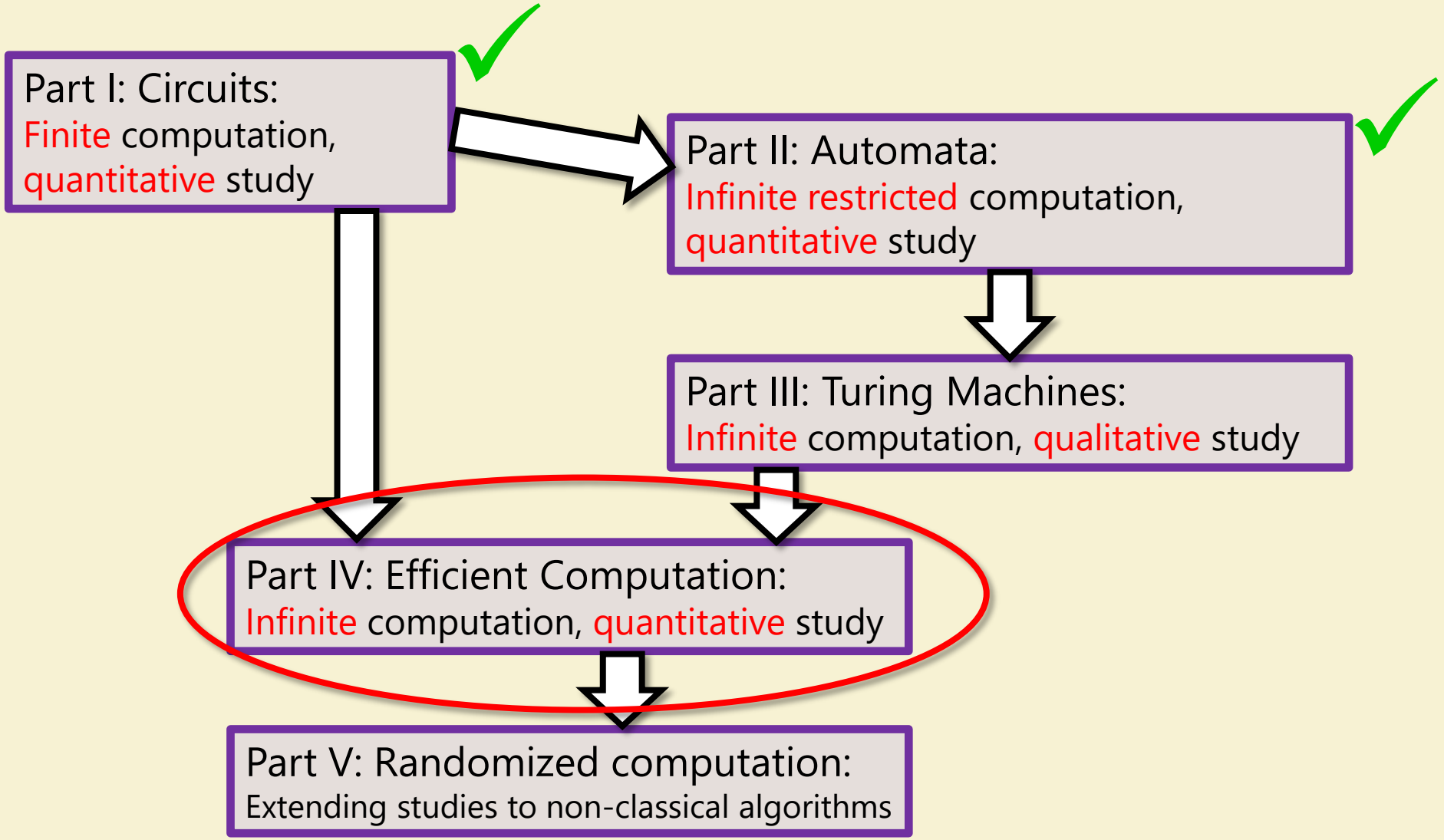
How to contact us {
- The whole staff (faster response): CS 121 Piazza
- Only the course heads (slower): cs121.fall2020.course.heads@gmail.com

Announcements:

- 121.5: Santhoshini Velusamy: Streaming Algorithms
- Sections: Midterm 2 review this week
- Homework 5 due in one week.

- General: CS Sophomore advising events:
  - Monday 10:30-11:30 & 4-5 (EST)

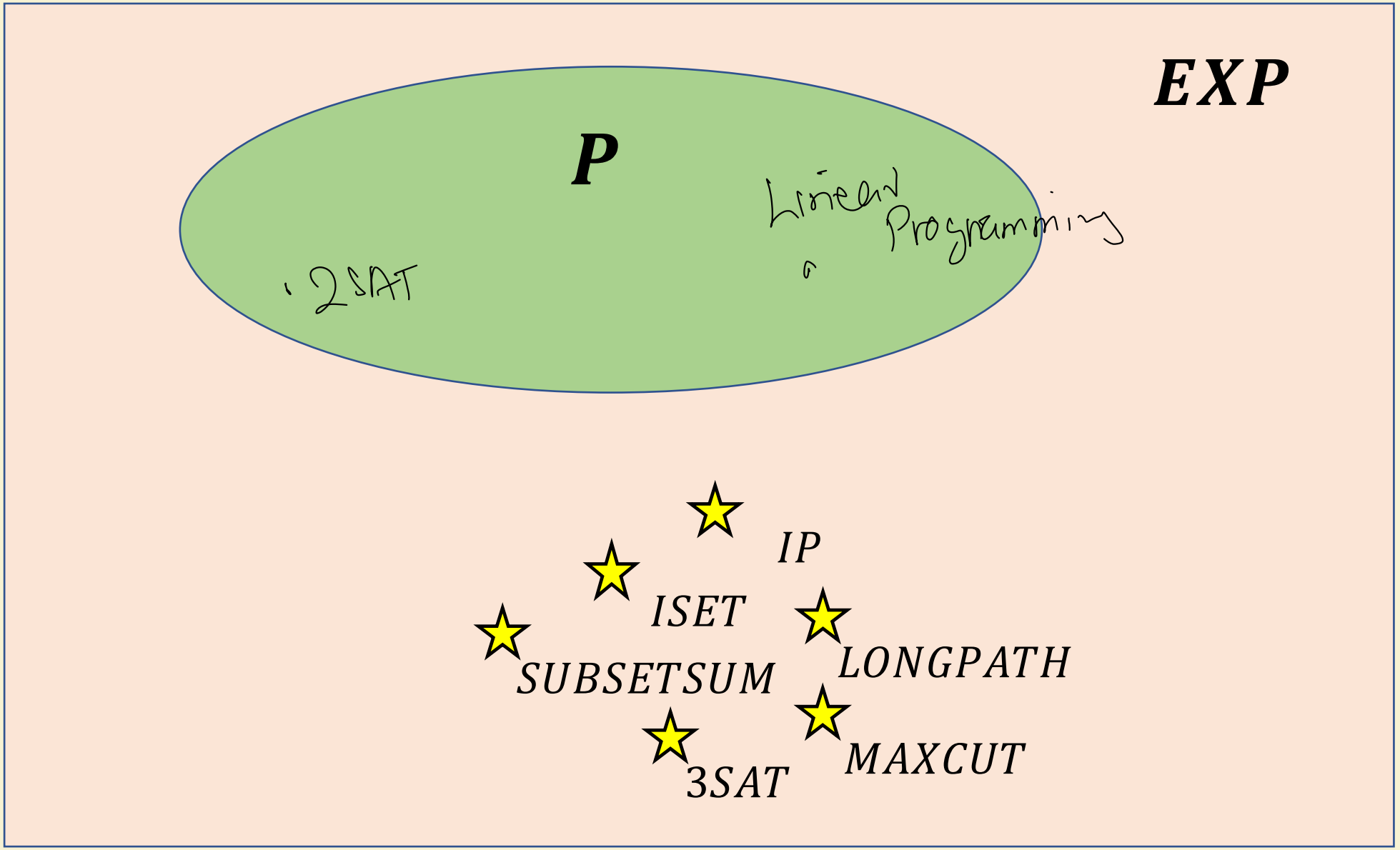
Where we are:


Part I: Circuits:
Finite computation, quantitative study
Part II: Automata:
Infinite restricted computation, quantitative study
Part III: Turing Machines:
Infinite computation, qualitative study
Part IV: Efficient Computation:
Infinite computation, quantitative study
Part V: Randomized computation:
Extending studies to non-classical algorithms

Review of last lecture

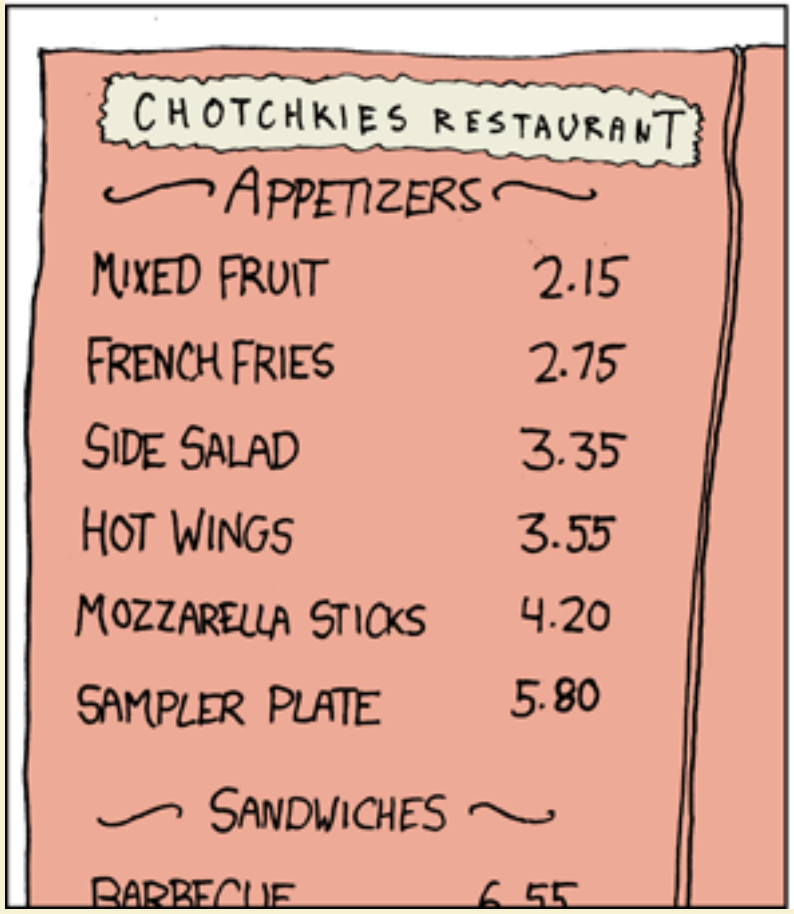
- Host of problems:
  - 2SAT, MinCut, LinearProgramming, (Shortest Path): All in polynomial time (Boolean versions in P).
  - 3SAT, MaxCut, IntegerProgramming, (Longest Path), ISET: All not known to be in polynomial time. (Boolean versions in EXP).
- Reductions: $F \leq_P G \Leftrightarrow \exists R$ such that $\forall x\ F(x) = G\big(R(x)\big), \quad R$ polytime.
  - 3SAT $\leq_P$ ISET
  - 3SAT $\leq_P$ IP (flashed)
  - MaxCut $\leq_P$ IP (flashed)

Previously


EXP
P
2SAT
Linear Programming
IP
ISET
SUBSETSUM
LONGPATH
3SAT
MAXCUT

MY HOBBY:

EMBEDDING NP-COMPLETE PROBLEMS IN RESTAURANT ORDERS



Today:

- What brings all the “hard” problems above together
  - “Verifiability”
- Definition of NP:
- How to show problems in NP
  - 3SAT in NP
  - MaxCut in NP
  - ISET in NP
- Definition of NP-Hardness and NP-Completeness:
- Some NP-complete problems …

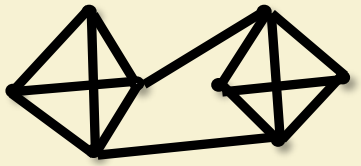
Verifying "Solutions"

- Claim: SAT, MaxCUT, ISET are example of functions $F:\{0,1\}^* \to \{0,1\}$ (defined/designed so that) the claim $F(x) = 1$ is "easy" to "verify".
- Example:
  - $\mathrm{SAT}((x_7 \vee \overline{x}_{17} \vee x_{29}) \wedge (\overline{x}_7 \vee x_{15} \vee x_{22}) \wedge (x_{22} \vee \overline{x}_{29} \vee x_{55})\,) = 1$

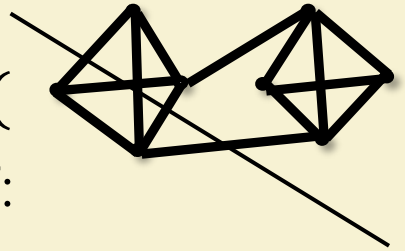
Verifying "Solutions"

- Claim: SAT, MaxCUT, ISET are example of functions $F:\{0,1\}^* \to \{0,1\}$ (defined/designed so that) the claim $F(x) = 1$ is "easy" to "verify".
- Example:
  - $\mathrm{SAT}((x_7 \vee \overline{x}_{17} \vee x_{29}) \wedge (\overline{x}_7 \vee x_{15} \vee x_{22}) \wedge (x_{22} \vee \overline{x}_{29} \vee x_{55})\,) = 1$
    - Proof: $x_7 = 1\,, x_{22} = 1$

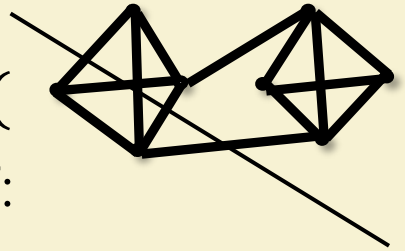
Verifying "Solutions"

- Claim: SAT, MaxCUT, ISET are example of functions $F:\{0,1\}^* \to \{0,1\}$ (defined/designed so that) the claim $F(x) = 1$ is "easy" to "verify".
- Example:
  - $\mathrm{SAT}((x_7 \vee \overline{x}_{17} \vee x_{29}) \wedge (\overline{x}_7 \vee x_{15} \vee x_{22}) \wedge (x_{22} \vee \overline{x}_{29} \vee x_{55})\,) = 1$
    - Proof: $x_7 = 1\,, x_{22} = 1$
  - $\mathrm{MaxCut}($  $, 4) = 1$

Verifying "Solutions"

- Claim: SAT, MaxCUT, ISET are example of functions $F:\{0,1\}^* \to \{0,1\}$ (defined/designed so that) the claim $F(x) = 1$ is "easy" to "verify".
- Example:
  - $\mathrm{SAT}((x_7 \vee \overline{x}_{17} \vee x_{29}) \wedge (\overline{x}_7 \vee x_{15} \vee x_{22}) \wedge (x_{22} \vee \overline{x}_{29} \vee x_{55})\,) = 1$
    - Proof: $x_7 = 1\,, x_{22} = 1$
  - $\mathrm{MaxCut}($  $, 4) = 1$
    - Proof:

Verifying "Solutions"

- Claim: SAT, MaxCUT, ISET are example of functions $F:\{0,1\}^* \to \{0,1\}$ (defined/designed so that) the claim $F(x) = 1$ is "easy" to "verify".
- Example:
  - $\mathrm{SAT}((x_7 \vee \overline{x}_{17} \vee x_{29}) \wedge (\overline{x}_7 \vee x_{15} \vee x_{22}) \wedge (x_{22} \vee \overline{x}_{29} \vee x_{55})\,) = 1$
    - Proof: $x_7 = 1\,, x_{22} = 1$
  - $\mathrm{MaxCut}($  $, 4) = 1$
    - Proof:
- But what do "Verify", "easy", "Proof" mean?

Formalizing "Verification"

- Input: String $x \in \{0,1\}^*$
- "Proof" : Another String $w \in \{0,1\}^*$ (aka "witness")
- "Verify" Verification Function $V_F: \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$
- "Easy": V_F computable in polynomial time (in first input length!)
- Putting it together: $F: \{0,1\}^* \to \{0,1\}$ easy to verify

if $\exists V_F: \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ s.t. $\forall x \in \{0,1\}^*$,

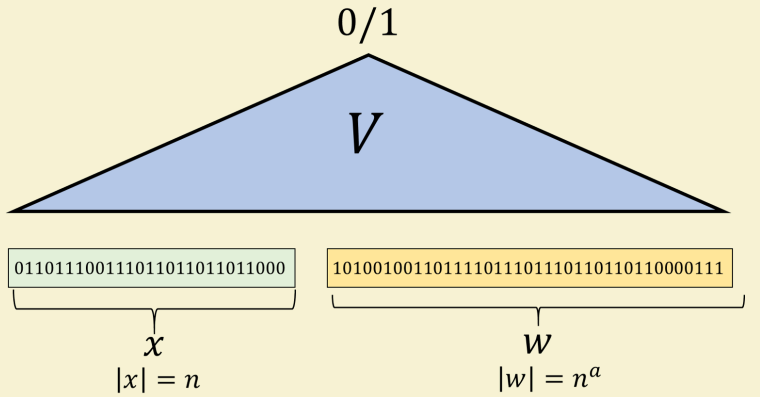
$$F(x) = 1 \Leftrightarrow \exists w \in \{0,1\}^* \text{ such that } V_F(x,w) = 1$$

and $V_F(x,w)$ computable in time $\text{poly}(|x|)$

$|w| \leq |x|^{O(1)}$ if $V_F(x,w) = 1$

Main definition of today

Def (informal): NP is the set of problems for which a solution can be efficiently verified.



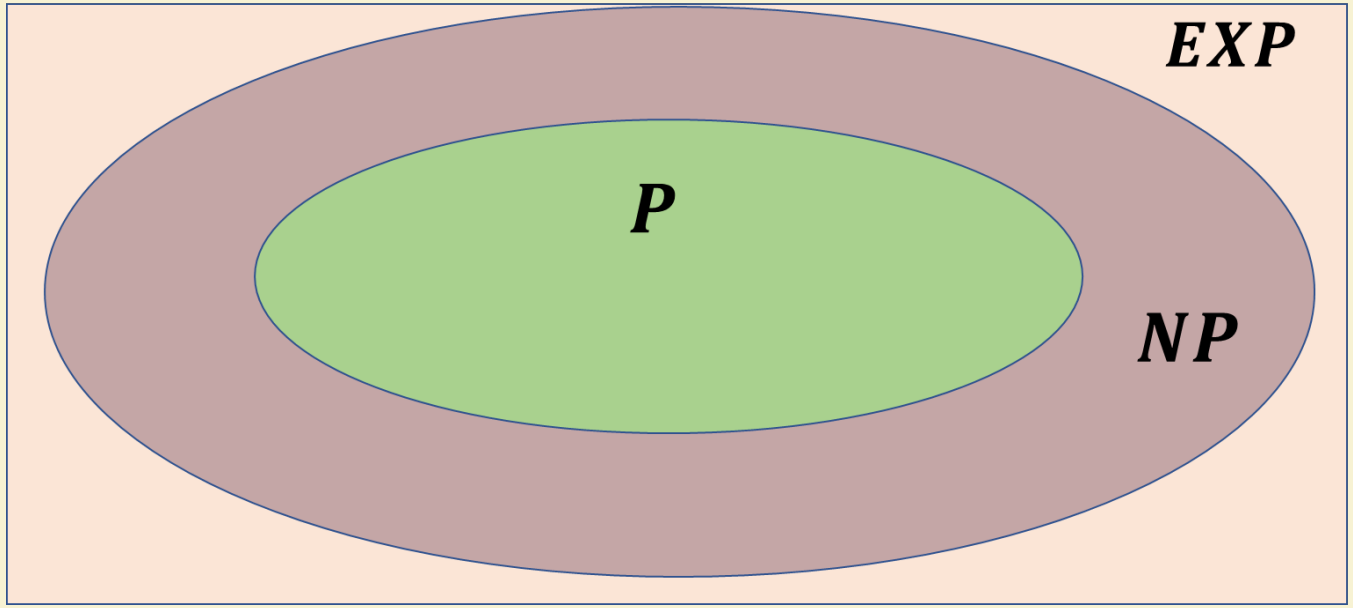
Example: 3SAT . It might be hard to find an satisfying assignment for given φ but given assignment x, we can verify that $\varphi(x) = 1$

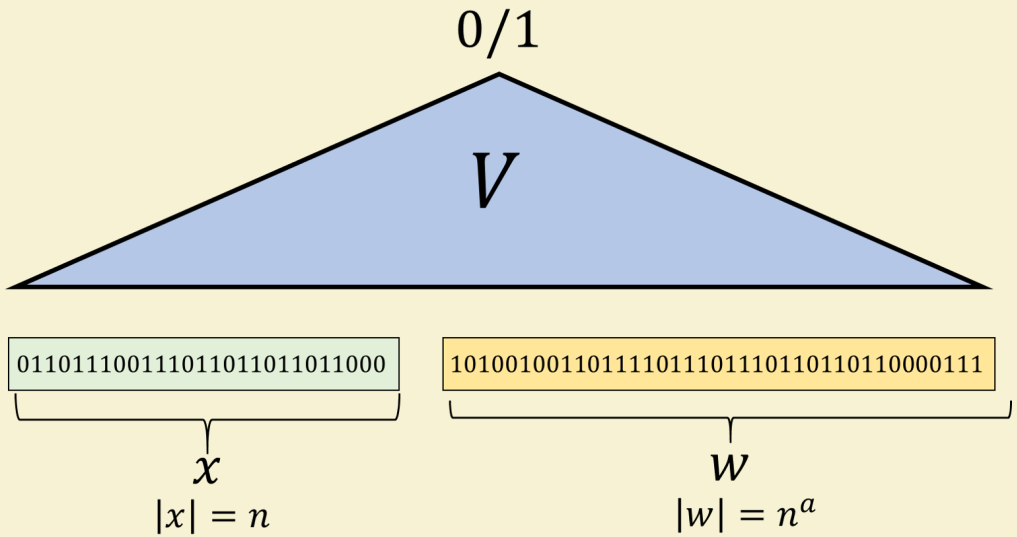
Slightly more formal: NP is set of functions $F: \{0,1\}^* \to \{0,1\}$ s.t. for every $x \in \{0,1\}^*$, $F(x) = 1$ if and only if there is some polynomial size "witness" / "solution" / "proof" demonstrating this.

Def: Let $F: \{0,1\}^* \to \{0,1\}$. Then $F \in NP$ if there is poly-time V_F and poly $q: \mathbb{N} \to \mathbb{N}$ such that for every $x \in \{0,1\}^*$

$$F(x) = 1 \Leftrightarrow \exists_{w \in \{0,1\}^*} |w| \leq q(|x|) \; and \; V_F(x,w) = 1$$

Main definition of today





$P \neq EXP$

$\Downarrow$

$\text{NOT}((P=NP) \text{ AND } (NP=EXP))$

Known: (1) $P \subseteq NP \subseteq EXP$ (2) $P \subsetneq EXP$

Unknown: Does $P = NP$? Does $NP = EXP$?

Central Open Question of CS: Does $P = NP$?

Example 1: $3SAT \in NP$

$(x_7 \vee \bar{x}_{22} \vee x_{17}) \wedge (\cdots) \cdots$

- Proof ingredients.
  - (Idea: What is w and what would verifier $V_{3\mathrm{SAT}}$ try to verify.)
  - The verifier $V_{3\mathrm{SAT}}$
  - Proof that $\forall \phi$ s.t. $3\mathrm{SAT}(\phi) = 1$ $\exists\, w$ s.t. $V_{3\mathrm{SAT}}(\phi, w) = 1$
  - $V_{3\mathrm{SAT}} \in P$ and $|w| = \mathrm{poly}(|\phi|)$

Example 1: $3SAT \in NP$

$C_j = x_{25} \vee \bar{x}_{76} \vee x_{30}$

- Proof ingredients.
  - (Idea: What is w and what would verifier $V_{3\text{SAT}}$ try to verify.)
  - The verifier $V_{3\text{SAT}}$
  - Proof that $\forall \phi$ s.t. $3\text{SAT}(\phi) = 1$ $\exists\, w$ s.t. $V_{3\text{SAT}}(\phi, w) = 1$
  - $V_{3\text{SAT}} \in P$ and $|w| = \text{poly}(|\phi|)$
- Actual Proof:
  - (Idea: w = assignment to variables $x_0 \ldots x_{n-1}$; Verifier verifies every clause satisfied.)

  - $V_{3\text{SAT}}(C_0 \wedge C_2 \cdots \wedge C_{m-1}, w) = 1 \Leftrightarrow \forall\, j \in [m-1], \geq 1$ literal in C_j is set to 1 by w
  - By definition $3\text{SAT}(\phi) = 1$ $\exists\, w$ s.t. $V_{3\text{SAT}}(\phi, w) = 1$.
  - $V_{3\text{SAT}} \in P$ and $|w| \leq |\phi|$

Example 2: $\mathbf{ISET \in NP}$

$V = \{0, 1, .. n-1\}$

- Recall $\text{ISET}(G = (V, E), k) = 1 \Leftrightarrow \exists S \subseteq V, |S| \geq k, \forall u, v \in S, (u, v) \notin E$
- Idea = ? $V_{\text{ISET}} = ?$

$w \in \{0,1\}^n$ = $w_i = 1 \Leftrightarrow i \in S$

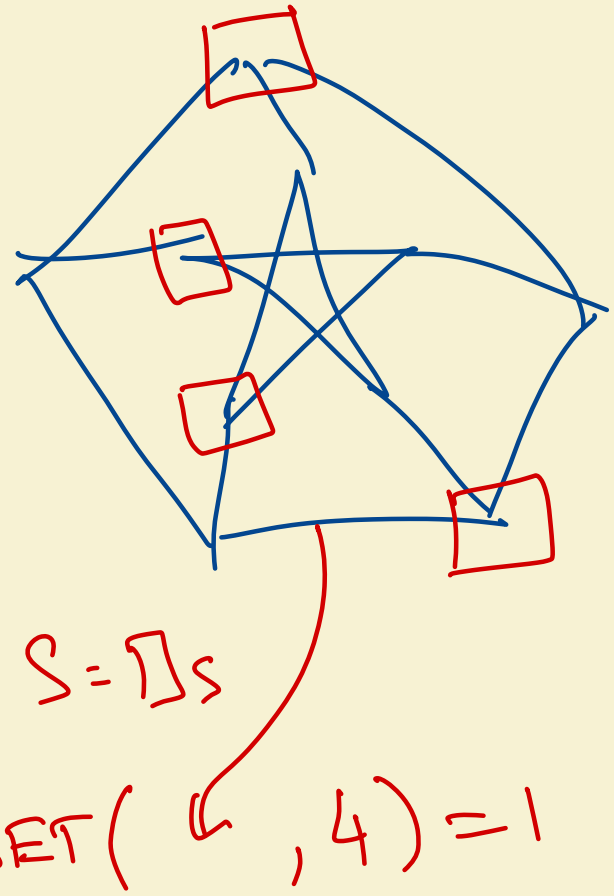
Alg-V_{ISET} $((G, k), w)$:

① if $w_0 + w_1 + \ldots w_{n-1} < k$ output 0.

② for every $(i, j) \in E$

if $w_i = w_j = 1$ output 0

③ Output 1

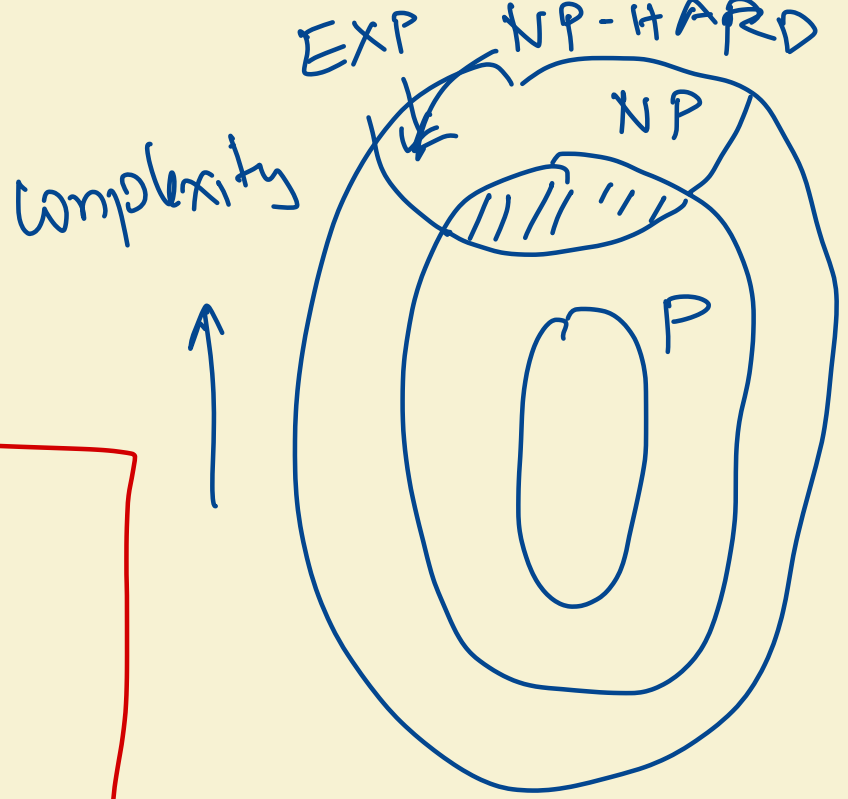


Example: $\mathbf{MaxCut} \in \mathbf{NP}$

- Recall: $\text{Maxcut}(G = (V, E), k) = 1 \Leftrightarrow \exists S \subseteq V \; s.t. \; |E \cap S \times S| \geq k$
  - (In English: vertices of G can be partitioned in two sets with $\geq k$ edges across partitions.)
- $\text{MaxCut} \in \text{NP}$? Idea = ? $V_{\text{MaxCut}} =$?

Exercise.

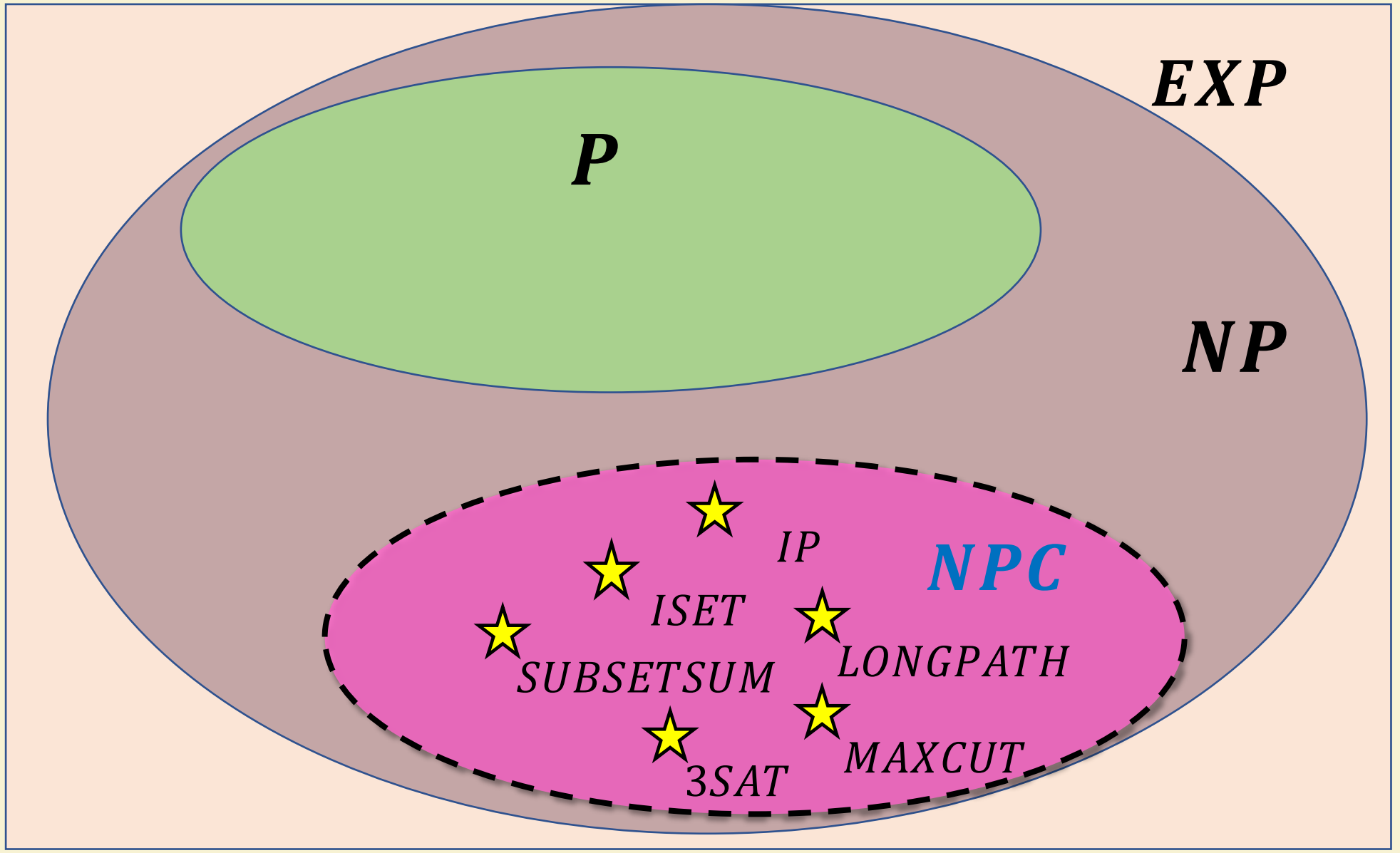
Preview of Next Lecture:



- **Cook/Levin Theorem**: 3SAT is NP-complete.
- F is **NP-hard** if $\forall G \in \text{NP}$, $G \leq_P F$
  - "every problem in NP reduces to F"
  - "No problem in NP harder than F"
- F is $\boldsymbol{NP}$**-complete** if $F \in NP$ and F is NP-hard

- Q: How many reductions needed to show 3SAT is NP-hard?

How many problems in NP?



Cook Levin Theorem implies: If $3SAT \in P$ then $P = NP$.

Cook Levin Theorem implies: If $\mathrm{MaxCut} \notin P$ then $3\mathrm{SAT} \notin P$.

Вы здесь?

ACM Symposium
Theory of Computing
May, 1971

Stephen A. Cook

University of Toronto

*66

Summary

It is shown that any recognition problem solved by a polynomial time-bounded nondeterministic Turing machine can be "reduced" to the problem of determining whether a given propositional formula is a tautology. Here "reduced" means, roughly speaking, that the first problem can be solved deterministically in polynomial time provided an oracle is available for solving the second. From this notion of reducible, polynomial degrees of difficulty are defined, and it is shown that the problem of determining tautologyhood has the same polynomial degree as the problem of determining whether the first of two given graphs is isomorphic to a subgraph of the second. Other examples are discussed. A method of measuring the complexity of proof procedures for the predicate calculus is introduced and discussed.

Throughout this paper, a set of strings means a set of strings on some fixed, large, finite alphabet Σ. This alphabet is large enough to include symbols for all sets described here. All Turing machines are deterministic recognition devices, unless the contrary is explicitly stated.

1. Tautologies and Polynomial Re-Reducibility.

Let us fix a formalism for the propositional calculus in which formulas are written as strings on Σ. Since we will require infinitely many proposition symbols (atoms), each such symbol will consist of a member of Σ followed by a number in binary notation to distinguish that symbol. Thus a formula of length n can only have about n/logn distinct function and predicate

certain recursive set of strings on this alphabet, and we are interested in the problem of finding a good lower bound on its possible recognition times. We provide no such lower bound here, but theorem 1 will give evidence that {tautologies} is a difficult set to recognize, since many apparently difficult problems can be reduced to determining tautologyhood. By reduced we mean, roughly speaking, that if tautologyhood could be decided instantly (by an "oracle") then these problems could be decided in polynomial time. In order to make this notion precise, we introduce query machines, which are like Turing machines with oracles in [1].

A query machine is a multitape Turing machine with a distinguished tape called the query tape, and three distinguished states called the query state, yes state, and no state, respectively. If M is a query machine and T is a set of strings, then a T-computation of M is a computation of M in which initially M is in the initial state and has an input string w on its input tape, and each time M assumes the query state there is a string u on the query tape, and the next state M assumes is the yes state if u∈T and the no state if u∉T. We think of an "oracle", which knows T, placing M in the yes state or no state.

Definition

A set S of strings is P-reducible (P for polynomial) to a set T of strings iff there is some query machine M and a polynomial Q(n) such that for each input string w, the T-computation of M with input w halts within Q(|w|) steps

Том IX 1973 Вып. 3

КРАТКИЕ СООБЩЕНИЯ

УДК 519.14

УНИВЕРСАЛЬНЫЕ ЗАДАЧИ ПЕРЕБОРА

Л. А. Левин

В статье рассматривается несколько известных массовых задач «переборного типа» и доказывается, что эти задачи можно решать лишь за такое время, за которое можно решать вообще любые задачи указанного типа.

После уточнения понятия алгоритма была доказана алгоритмическая неразрешимость ряда классических массовых проблем (например, проблем тождества элементов групп, гомеоморфности многообразий, разрешимости диофантовых уравнений и других). Тем самым был снят вопрос о нахождении практического способа их решения. Однако существование алгоритмов для решения других задач не снимает для них аналогичного вопроса из-за фантастически большого объема работы, предписываемого этими алгоритмами. Такова ситуация с так называемыми переборными задачами: минимизации булевых функций, поиска доказательств ограниченной длины, выяснения изоморфности графов и другими. Все эти задачи решаются тривиальными алгоритмами, состоящими в переборе всех возможностей. Однако эти алгоритмы требуют экспоненциального времени работы и у математиков сложилось убеждение, что более простые алгоритмы для них невозможны. Был получен ряд серьезных аргументов в пользу его справедливости (см.[1, 2]), однако доказать это утверждение не удалось никому. (Например, до сих пор не доказано, что для нахождения математических доказательств нужно больше времени, чем для их проверки.)

Однако если предположить, что вообще существует какая-нибудь (хотя бы искусственно построенная) массовая задача переборного типа, неразрешимая простыми (в смысле объема вычислений) алгоритмами, то можно показать, что этим же свойством обладают и многие «классические» переборные задачи (в том числе задача минимизации, задача поиска доказательств и др.). В этом и состоят основные результаты статьи.

Функции $f(n)$ и $g(n)$ будем называть сравнимыми, если при некотором k

$$f(n) \leqslant (g(n)+2)^k \quad \text{и} \quad g(n) \leqslant (f(n)+2)^k.$$

Аналогично будем понимать термин «меньше или сравнимо».

Определение. Задачей переборного типа (или просто переборной задачей) будем называть задачу вида «по данному x найти какое-нибудь y длины, сравнимой

REDUCIBILITY AMONG COMBINATORIAL PROBLEMS†

Richard M. Karp *59

University of California at Berkeley

Conf on 1972

Abstract: A large class of computational problems involve determination of properties of graphs, digraphs, integers, of integers, finite families of finite sets, boolean formul elements of other countable domains. Through simple encodi from such domains into the set of words over a finite alpha these problems can be converted into language recognition p and we can inquire into their computational complexity. It reasonable to consider such a problem satisfactorily solved an algorithm for its solution is found which terminates wit number of steps bounded by a polynomial in the length of th We show that a large number of classic unsolved problems of ing, matching, packing, routing, assignment and sequencing equivalent, in the sense that either each of them possesses polynomial-bounded algorithm or none of them does.

1. INTRODUCTION

All the general methods presently known for computing chromatic number of a graph, deciding whether a graph has a Hamilton circuit, or solving a system of linear inequalitie which the variables are constrained to be 0 or 1, requi combinatorial search for which the worst case time requirem grows exponentially with the length of the input. In this paper we give theorems which strongly suggest, but do not imply, that these problems, as well as many others, will remain intractable

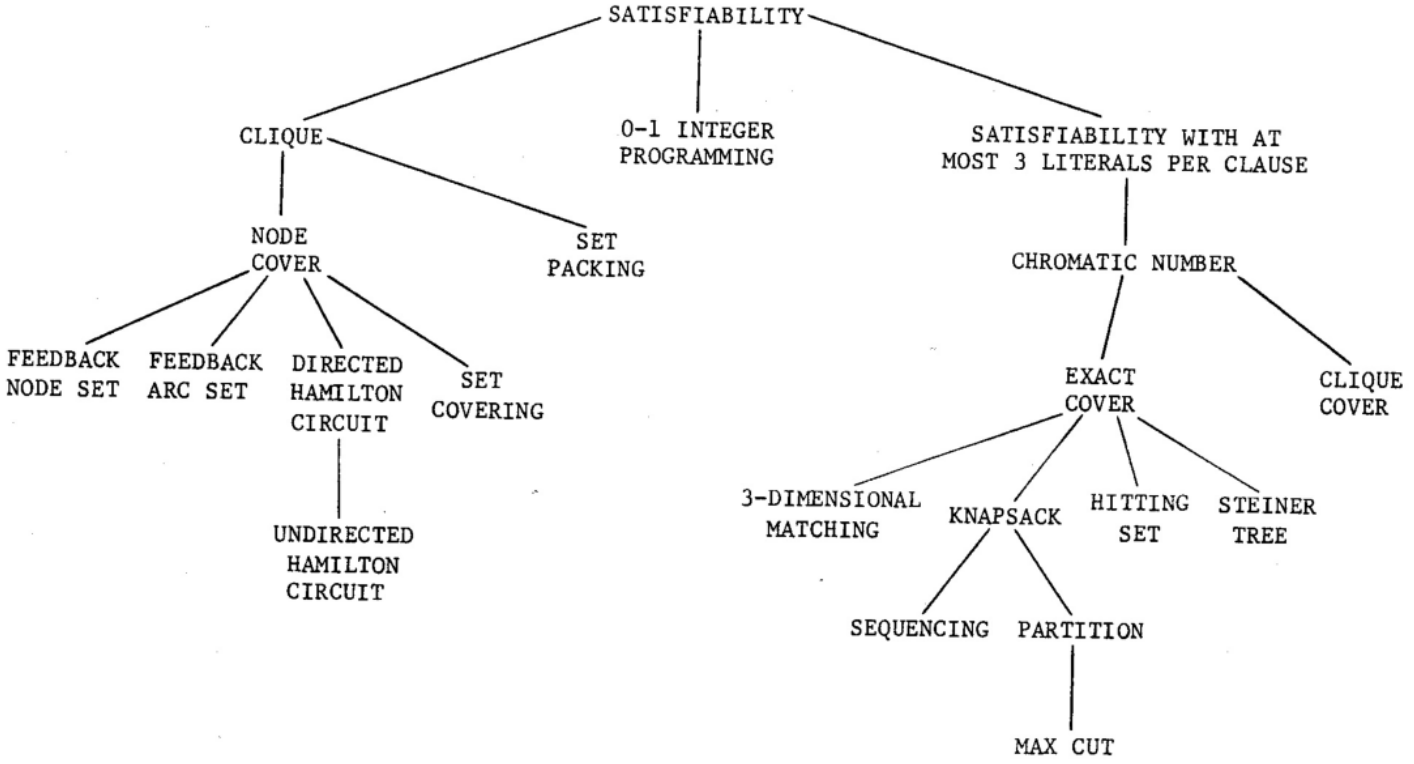
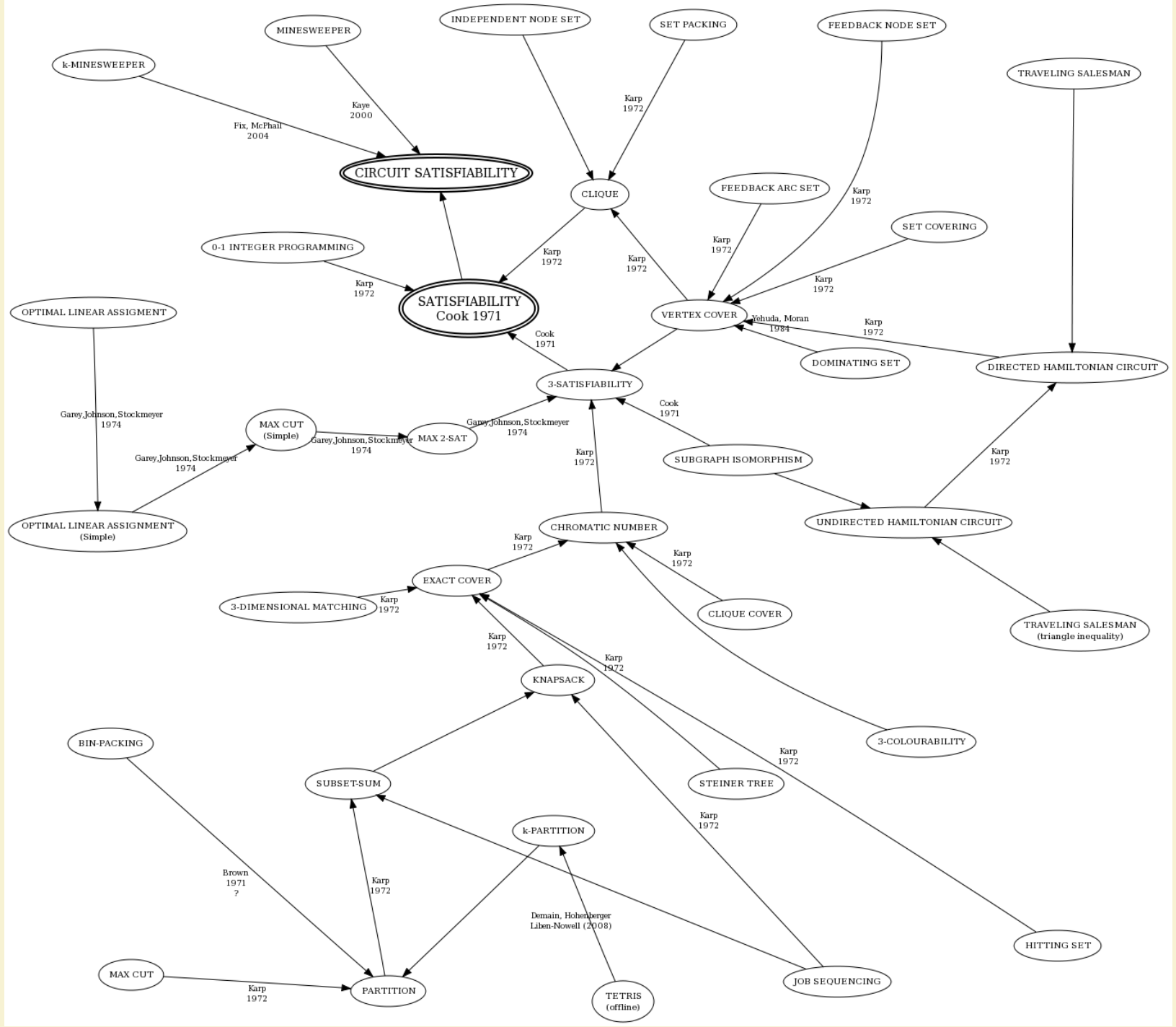


FIGURE 1 - Complete Problems



gradient_dissent / reddit

Assuming Cook/Levin: **ISET** is **NP**-complete

- Proof:
  1. $\text{ISET} \in \text{NP}$ – already done!
  2. $\text{3SAT} \leq_P \text{ISET}$ – already done!
  3. …

3SAT is NP-hard $\Rightarrow$ $\forall G \in NP$ $G \leq_p 3SAT$

$G \leq_p 3SAT \leq_p ISET$

$G \leq_p ISET$

(needs Proof)

(good exercise)

X is NP-complete

$X \in NP$; find Y NP-hard

$Y \leq_p X$

Assuming Cook/Levin: **ISET** is **NP**-complete

- Proof:
  1. $\mathrm{ISET} \in \mathrm{NP}$ – already done!
  2. $3\mathrm{SAT} \leq_P \mathrm{ISET}$ – already done!
  3. $\forall F \in \mathrm{NP}, F \leq_P \mathrm{SAT} \leq_P \mathrm{ISET} \quad \Rightarrow \quad F \leq_P \mathrm{ISET}.$ QED

Summary of Lecture:

- Introduced NP
  - Class of "verifiable" problems. (Warning: Only $F(x) = 1$ verifiable.)
  - Example problems in NP: 3SAT, MaxCUT, ISET
- Introduced NP-hard and NP-complete
  - Stated Cook/Levin Theorem (proof next lecture).
  - Implications of Cook-Levin
    - 3SAT in P $\Leftrightarrow$ P=NP
    - $\text{MaxCut} \notin P \Rightarrow \text{3SAT} \notin \text{NP}$
    - 3SAT NP-complete $\Rightarrow$ ISET NP-complete

Summary of Lecture:

- Introduced NP
  - Class of "verifiable" problems. (Warning: Only $F(x) = 1$ verifiable.)
  - Example problems in NP: 3SAT, MaxCUT, ISET
- Introduced NP-hard and NP-complete
  - Stated Cook/Levin Theorem (proof next lecture).
  - Implications of Cook-Levin
    - 3SAT in P $\Leftrightarrow$ P=NP
    - $\text{MaxCut} \notin P \Rightarrow \text{3SAT} \notin \text{NP}$
    - 3SAT NP-complete $\Rightarrow$ ISET NP-complete