# CS630 Graduate Algorithms

September 17, 2024

by Dora Erdos and Jeffrey Considine

- Certificates
- NP problems
- NP-complete problems

slides by Dora Erdos and Jeffrey Considine, based on Kevin Wayne, Adam Smith

# Polynomial-time reductions

Suppose there is an algorithm to solve problem Y.

- the algorithm is unknown to us, we treat it as a *black-box* or *API*

Polynomial Reduction. Problem $X$ is polynomial-time reducible to problem $Y$ if arbitrary instances of problem $X$ can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to an oracle/black box that solves problem $Y$.

Notation. $X \leq_P Y$.

Bipartite Matching $\leq_p$ Max Flow

- define flow graph corresponding to the bipartite graph
- find max flow (using FF)
- iterate over middle edges and assign pairs with non-zero flow to the matching

# Factoring

Does integer $n$ have any non-trivial factors?

- Integers $p$, $q$ such that $pq = n$, $1 < p < n$

# Factoring

Does integer $n$ have any non-trivial factors?

- Integers $p$, $q$ such that $pq = n$, $1 < p < n$

This is an important operation for cryptography.

- Many early public key cryptography systems assumed that finding factors of large numbers is hard (RSA, DSA).
  - Specifically finding factors, not deciding whether there are any.
  - Still used a lot today.

# Factoring

## Does integer $n$ have any non-trivial factors?

- Integers $p$, $q$ such that $pq = n$, $1 < p < n$

- Try $n = 883229031130271638468553200480417782229$

# Factoring

Does integer $n$ have any non-trivial factors?

- Integers $p$, $q$ such that $pq = n$, $1 < p < n$

- Try $n = 883229031130271638468553200480417782229$

There is a fast algorithm to check this.

- PRIMES is in P (AKS 2004)
- Problem size is $\log n$.
- Original algorithm took $\tilde{O}((\log n)^{12})$, later reduced to $\tilde{O}((\log n)^{6})$
  - Õ() ignores some logarithmically smaller factors…

# Factoring

Does integer $n$ have any non-trivial factors?

- Integers $p$, $q$ such that $pq = n$, $1 < p < n$
- Try $n = 8832290311302716384685532004804177822229$

There is a fast algorithm to check this.

- PRIMES is in P (AKS 2004)
- Problem size is $\log n$.
- Original algorithm took Õ$((\log n)^{12})$, later reduced to Õ$((\log n)^{6})$
  - Õ() ignores some logarithmically smaller factors…

This proof uses a bunch of number theory that I don't know yet.

- Can you quickly convince me that a number has non-trivial factors some other way?

# Factoring

Does integer $n$ have any non-trivial factors?

- Integers $p$, $q$ such that $pq = n$, $1 < p < n$

- Try $n = 8832290311302716384685532004804177822229$

You can prove that the answer is yes by providing a non-trivial factor $p$.

- Given $n$ and $p$, divide $n$ by $n$ and check if there is a remainder.
- If the remainder is zero, then yes, $n$ has a non-trivial factor.

# Factoring

Does integer $n$ have any non-trivial factors?

- Integers $p$, $q$ such that $pq = n$, $1 < p < n$

- Try $n = 883229031130271638468553200480417782229$

You can prove that the answer is yes by providing a non-trivial factor $p$.

- Given $n$ and $p$, divide $n$ by $n$ and check if there is a remainder.
- If the remainder is zero, then yes, $n$ has a non-trivial factor.

- $p = 37040716014851403527$

- $883229031130271638468553200480417782229 \equiv 0 \pmod{23844815277764681027}$

# Factoring

Does integer $n$ have any non-trivial factors?

- Integers $p$, $q$ such that $pq = n$, $1 < p < n$

- Try $n = 8832290311302716384685532004804177 82229$

You can prove that the answer is yes by providing a non-trivial factor $p$.

- Given $n$ and $p$, divide $n$ by $n$ and check if there is a remainder.
- If the remainder is zero, then yes, $n$ has a non-trivial factor.

Using $p$ to certify that n has non-trivial factors is a lot faster than the AKS test.

- $O((\log n)^2)$ time using long division
- Much faster than Õ$((\log n)^6)$ of AKS.

# Factoring

Does integer $n$ have any non-trivial factors?

- Integers $p$, $q$ such that $pq = n$, $1 < p < n$

- Try $n = 88322903113027163846855320048041778222 9$

You can prove that the answer is yes by providing a non-trivial factor $p$.

- Given $n$ and $p$, divide $n$ by $n$ and check if there is a remainder.
- If the remainder is zero, then yes, $n$ has a non-trivial factor.

Using $p$ to certify that n has non-trivial factors is a lot faster than the AKS test.

- $O((\log n)^2)$ time using long division
- Much faster than Õ$((\log n)^6)$ of AKS.

But how do we get $p$?

# Factoring Takeaways

# Factoring Takeaways

## Certificates

- Make checking “yes” answers to decision problems easy.

# Factoring Takeaways

## Certificates

- Make checking “yes” answers to decision problems easy.
  - Providing $p$ allows quick confirmation of non-trivial factors.

# Factoring Takeaways

## Certificates

- Make checking “yes” answers to decision problems easy.
  - Providing $p$ allows quick confirmation of non-trivial factors.
  - A polynomial time certificate lets us confirm a yes answer to a decision problem in polynomial time.

# Top Hat Question

Which of the following is not an appropriate polynomial time certificate?

| | |
|---|---|
| A | A minimum cut for the maximum flow problem |
| B | A non-trivial factor for the factoring problem |
| C | A path for the graph reachability problem |
| D | Black or white to win for a chess problem |

# Longest Path and Hamiltonian Path problems

Longest Path:

- Given a weighted graph G and an integer k, is there a simple path of lengths at least k?

Hamiltonian-path problem:

- Given an unweighted graph G(V,E) is there a simple path that contains every node exactly once?

We do not know fast algorithms for these problems.

Do they have polynomial time certificates?

# Longest Path and Hamiltonian Path problems

Longest Path:

- Given a weighted graph G and an integer k, is there a simple path of lengths at least k?

Hamiltonian-path problem:

- Given an unweighted graph G(V,E) is there a simple path that contains every node exactly once?

We do not know fast algorithms for these problems.

Do they have polynomial time certificates?

- Yes! The paths are polynomial time certificates for yes answers.

# NP Problems

The class $NP$ is the class of problems with polynomial-time certificates.

# NP Problems

The class $NP$ is the class of problems with polynomial-time certificates.

- Many equivalent definitions of $NP$.
- We will focus on this certificate version.

# NP Problems

The class $NP$ is the class of problems with polynomial-time certificates.

- Many equivalent definitions of $NP$.
- We will focus on this certificate version.

All problems that can be solved in polynomial time are in $NP$.

# NP Problems

The class $NP$ is the class of problems with polynomial-time certificates.

- Many equivalent definitions of $NP$.
- We will focus on this certificate version.

All problems that can be solved in polynomial time are in $NP$.

- $P \subseteq NP$

# NP Problems

The class $NP$ is the class of problems with polynomial-time certificates.

- Many equivalent definitions of $NP$.
- We will focus on this certificate version.

All problems that can be solved in polynomial time are in $NP$.

- $P \subseteq NP$
- Use a blank certificate and run the polynomial time algorithm.

# NP Problems

The class $NP$ is the class of problems with polynomial-time certificates.

- Many equivalent definitions of $NP$.
- We will focus on this certificate version.

All problems that can be solved in polynomial time are in $NP$.

- $P \subseteq NP$
- Use a blank certificate and run the polynomial time algorithm.
- Some certificates will enable faster verification.
  - Saw that with factoring.

# NP Problems

The class $NP$ is the class of problems with polynomial-time certificates.

- Many equivalent definitions of $NP$.
- We will focus on this certificate version.

All problems that can be solved in polynomial time are in $NP$.

- $P \subseteq NP$
- Use a blank certificate and run the polynomial time algorithm.
- Some certificates will enable faster verification.
  - Saw that with factoring.

What problems are in NP but are not known to be in P?

# NP-Completeness

A problem $Y$ is $NP$-complete if every $NP$ problem $X$ can be reduced to $Y$ in polynomial time.

- Equivalently, $(Y \in NPC) \Leftrightarrow (\forall X \in NP, X \leq_P Y)$

# NP-Completeness

A problem $Y$ is $NP$-complete if every $NP$ problem $X$ can be reduced to $Y$ in polynomial time.

- Equivalently, $(Y \in NPC) \Leftrightarrow (\forall X \in NP, X \leq_P Y)$

$NP$-complete are the hardest known problems in $NP$.

- If any $NP$ problem is hard, all $NP$-complete problems are hard.
- If any $NP$ problem requires super-polynomial time, all $NP$-complete problems require super-polynomial time.

# NP-Completeness

A problem $Y$ is $NP$-complete if every $NP$ problem $X$ can be reduced to $Y$ in polynomial time.

- Equivalently, $(Y \in NPC) \Leftrightarrow (\forall X \in NP, X \leq_P Y)$

$NP$-complete are the hardest known problems in $NP$.

- If any $NP$ problem is hard, all $NP$-complete problems are hard.
- If any $NP$ problem requires super-polynomial time, all $NP$-complete problems require super-polynomial time.

- If any $NP$-complete problem can be solved in polynomial time, all $NP$ problems can be solved in polynomial time.

# NP-Completeness

A problem $Y$ is $NP$-complete if every $NP$ problem $X$ can be reduced to $Y$ in polynomial time.

- Equivalently, $(Y \in NPC) \Leftrightarrow (\forall X \in NP, X \leq_P Y)$

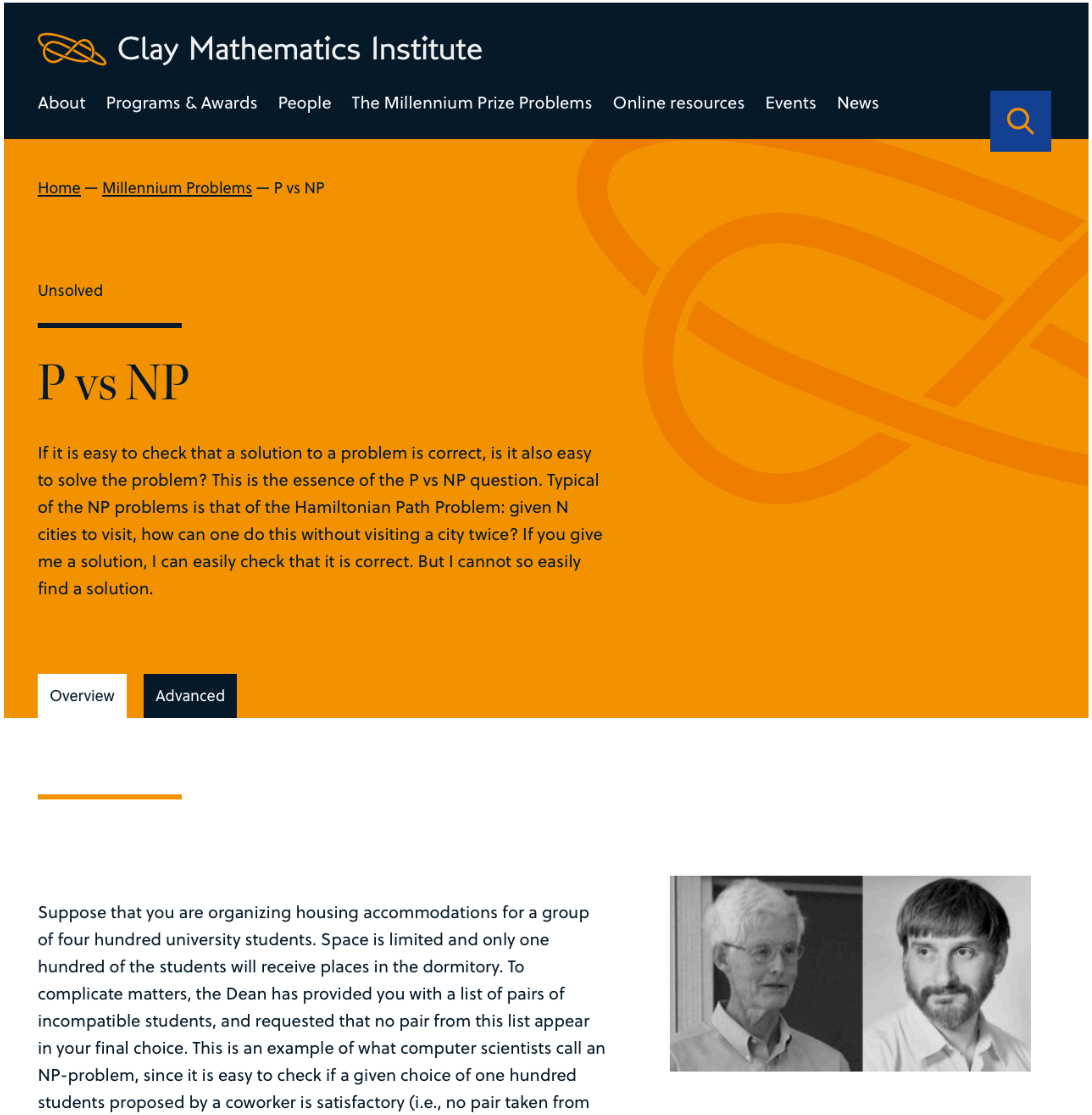
$NP$-complete are the hardest known problems in $NP$.

- If any $NP$ problem is hard, all $NP$-complete problems are hard.
- If any $NP$ problem requires super-polynomial time, all $NP$-complete problems require super-polynomial time.

- If any $NP$-complete problem can be solved in polynomial time, all $NP$ problems can be solved in polynomial time.

No such results so far.

# Millenium Prize Problem - $1,000,000 if you solve P=NP



Clay Mathematics Institute

About | Programs & Awards | People | The Millennium Prize Problems | Online resources | Events | News

Home — Millennium Problems — P vs NP

Unsolved

# P vs NP

If it is easy to check that a solution to a problem is correct, is it also easy to solve the problem? This is the essence of the P vs NP question. Typical of the NP problems is that of the Hamiltonian Path Problem: given N cities to visit, how can one do this without visiting a city twice? If you give me a solution, I can easily check that it is correct. But I cannot so easily find a solution.

Overview | Advanced

Suppose that you are organizing housing accommodations for a group of four hundred university students. Space is limited and only one hundred of the students will receive places in the dormitory. To complicate matters, the Dean has provided you with a list of pairs of incompatible students, and requested that no pair from this list appear in your final choice. This is an example of what computer scientists call an NP-problem, since it is easy to check if a given choice of one hundred students proposed by a coworker is satisfactory (i.e., no pair taken from

# *NP*-Complete Problems

Satisfiability (Cook 1971) and Tiling (Levin 1973)

Both took a similar approach

- Encode the execution of a program in minute detail.
  - Cook used boolean variables.
  - Levin used Turing machine states.
- Setup a process to confirm that the rules were followed.
  - Boolean formulas checking one step properly execute.
  - Tiles that only match valid Turing machine transitions.
- Check if there is a complete solution subject to those constraints.

# Karp's 21 NP-complete problems

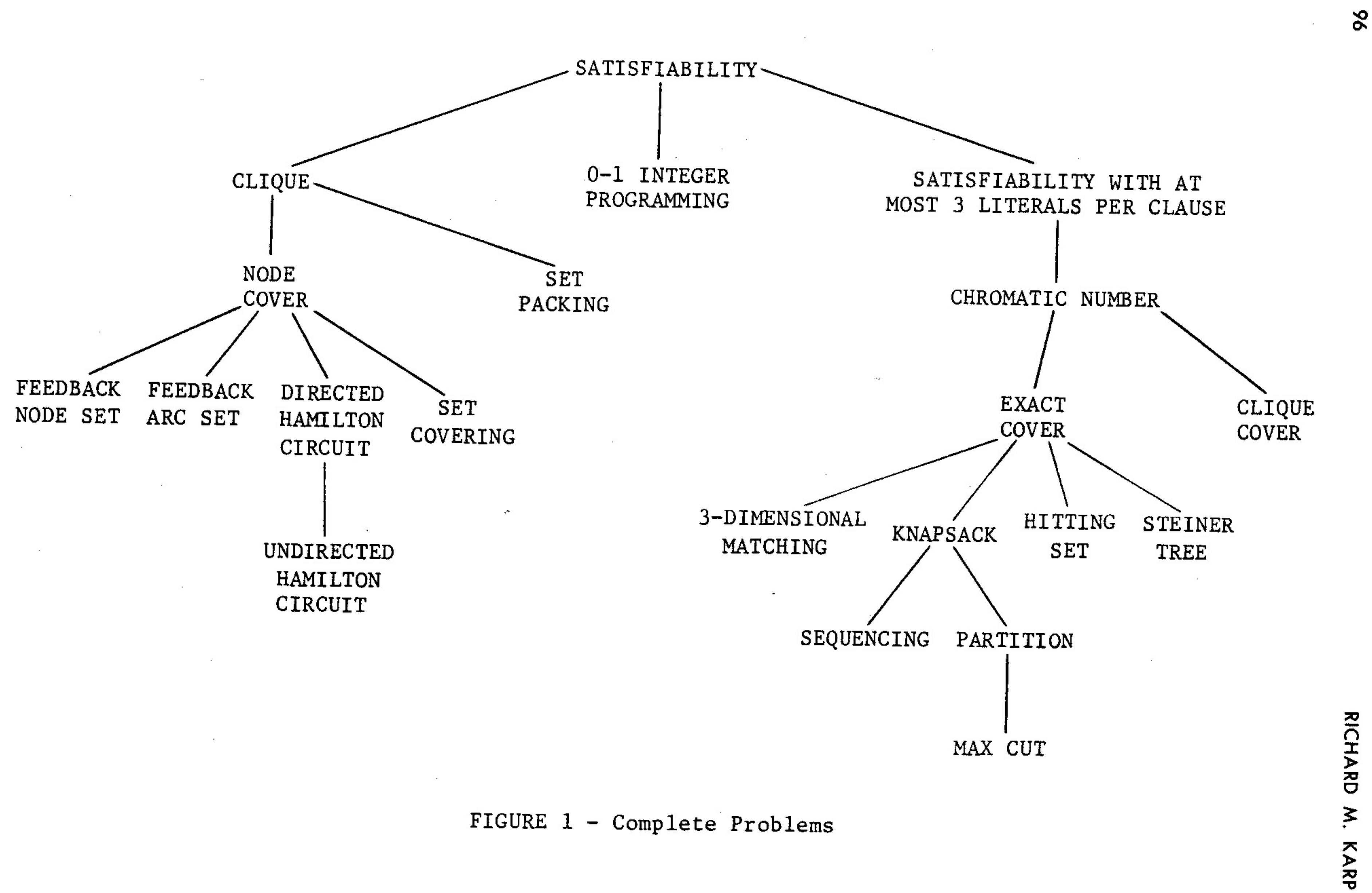


FIGURE 1 - Complete Problems

Image source: "Reducability Among Combinatorial Problems" (Karp 1972)

# Example $NP$-Complete Reductions

Will sketch out this chain of reductions

- SATISFIABILITY $\leq_P$ 3SAT
- 3SAT $\leq_P$ CLIQUE
- CLIQUE $\leq_P$ INDEPENDENT SET
- INDEPENDENT SET $\leq_P$ VERTEX COVER
- VERTEX COVER $\leq_P$ SET COVER

# Example $NP$-Complete Reductions

Will sketch out this chain of reductions

- SATISFIABILITY $\leq_P$ 3SAT
- 3SAT $\leq_P$ CLIQUE
- CLIQUE $\leq_P$ INDEPENDENT SET
- INDEPENDENT SET $\leq_P$ VERTEX COVER
- VERTEX COVER $\leq_P$ SET COVER

These are transitive, so proving these proves SATISFIABILITY $\leq_P$ SET COVER.

# Example $NP$-Complete Reductions

Will sketch out this chain of reductions

- SATISFIABILITY $\leq_P$ 3SAT
- 3SAT $\leq_P$ CLIQUE
- CLIQUE $\leq_P$ INDEPENDENT SET
- INDEPENDENT SET $\leq_P$ VERTEX COVER
- VERTEX COVER $\leq_P$ SET COVER

These are transitive, so proving these proves SATISFIABILITY $\leq_P$ SET COVER.

These problems are all in $NP$, so all of them are $NP$-complete.

# SATISFIABILITY $\leq_P$ 3SAT

How to map arbitrary Boolean formulas to this format?

- $(v_1 \lor v_2 \lor \neg v_3) \land (v2 \lor \neg v2 \lor v_4) \land \ldots$

- Conjunction (AND) of clauses of 3 literals.
- A literal is a variable or its negation.

Reduction trick:

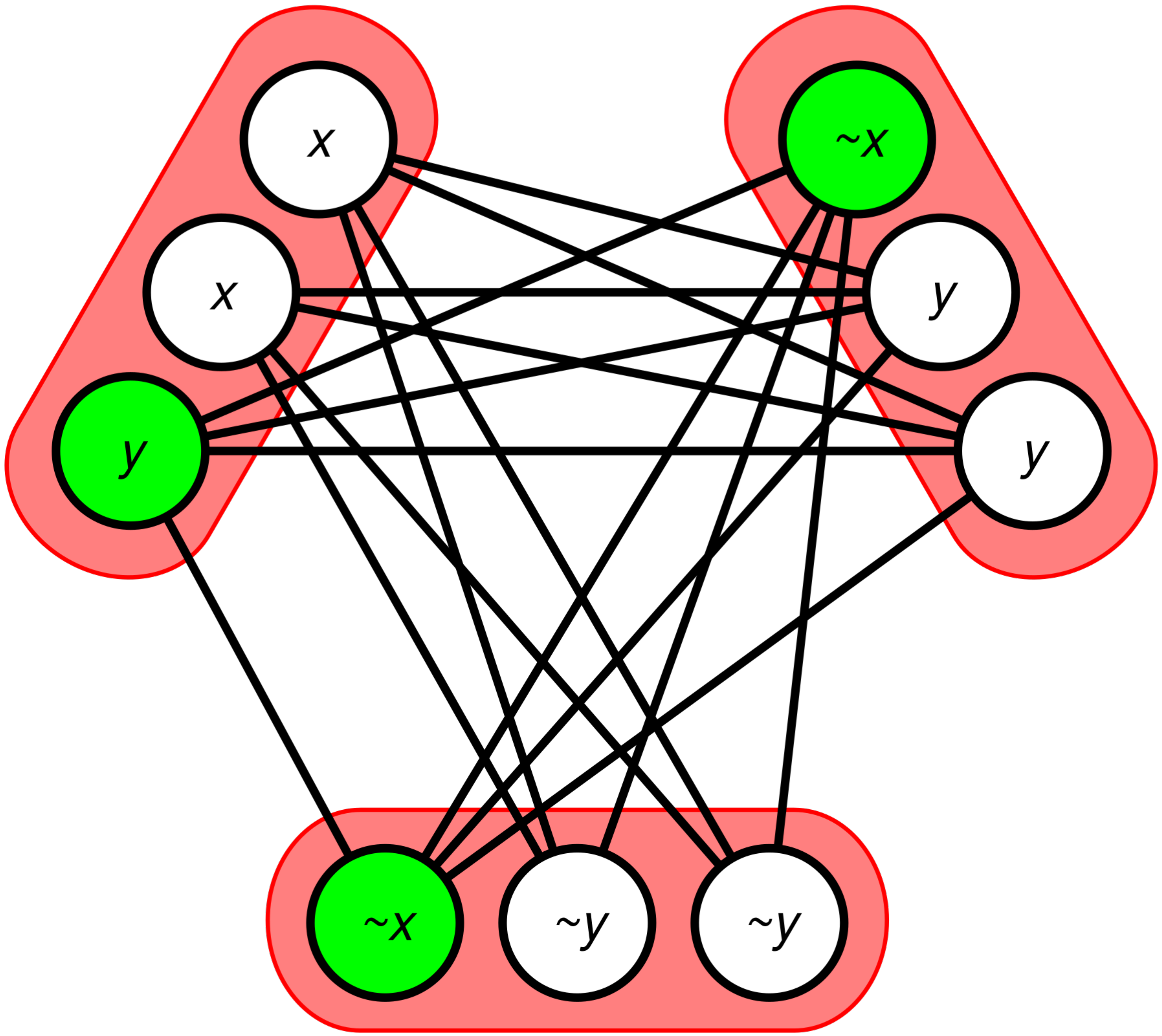
- Create new variable for every sub-expression in the SATISFIABILITY problem.
- Write clauses constraining sub-expression variables to their right values.
  - NOT WIDGET: $v_1 = \neg v_2 \Leftrightarrow (\neg v_1 \lor \neg v_2) \land (v_1 \lor v_2)$
  - AND WIDGET:
    $v_1 = (v_2 \land v_3) \Leftrightarrow (v_2 \lor \neg v_1) \land (v_3 \lor \neg v_1) \lor (v_1 \lor \neg v_2 \lor \neg v_3)$
  - OR WIDGET:
    $v_1 = (v_2 \lor v_3) \Leftrightarrow (\neg v2 \lor v_1) \land (\neg v3 \lor v_1) \land (\neg v_1 \lor v_2 \lor v_3)$

- Some of these just use 2 literals, but easy to pad to 3.

# 3SAT $\leq_P$ CLIQUE

## CLIQUE problem:

- Given a graph $G$, is there a subset of $k$ nodes that are fully connected?




- Image source: Wikipedia

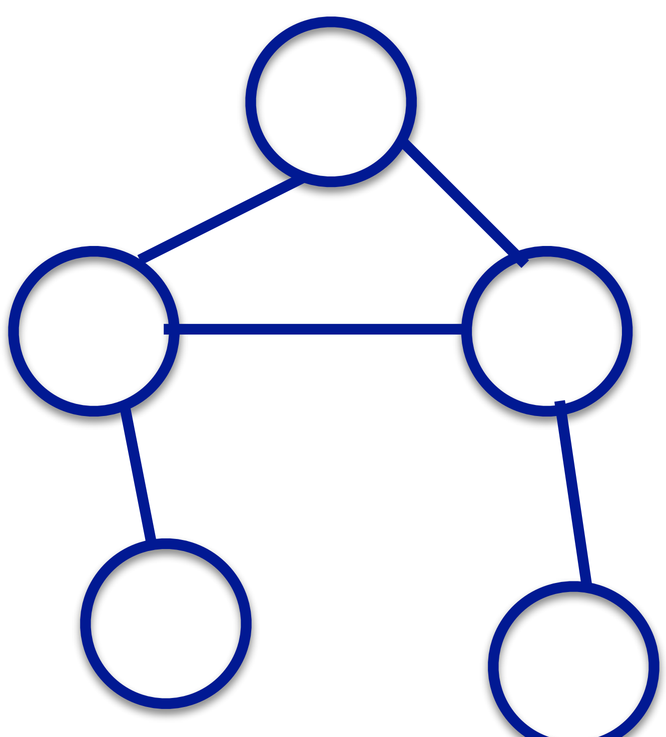
# CLIQUE $\leq_P$ INDEPENDENT SET

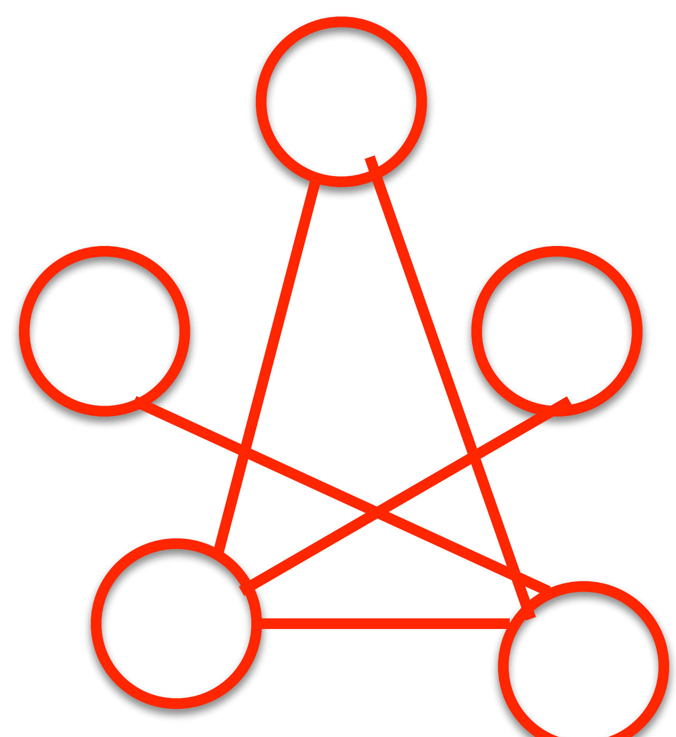
## INDEPENDENT SET problem

- Given a graph $G$, is there a subset of $k$ vertices with no edges between them?

## CLIQUE reduction

- Both problems are looking for a subset of $k$ vertices.
  - CLIQUE wants all the edges to be present between these vertices.
  - INDEPENDENT SET wants none of those edges to be present.
- Flip all the edges of $G$ and query INDEPENDENT SET with the same $k$.





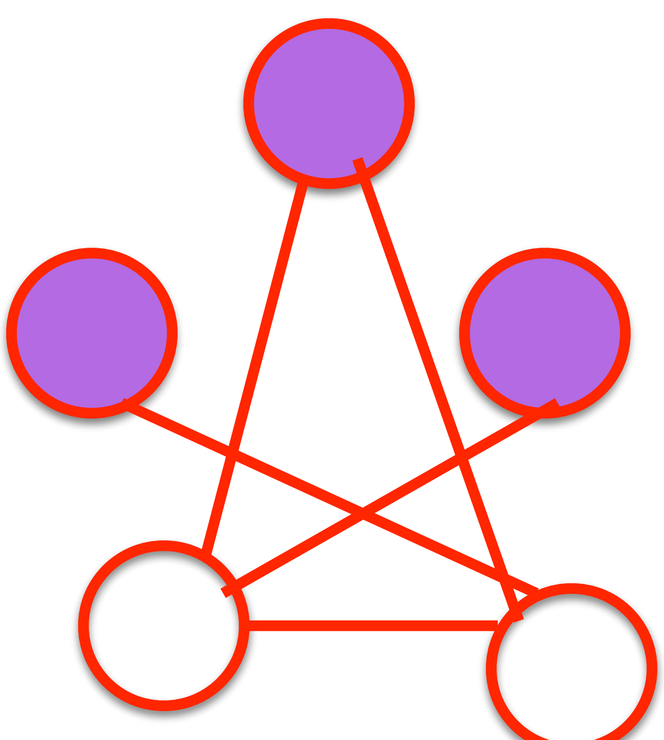
# INDEPENDENT SET $\leq_P$ VERTEX COVER

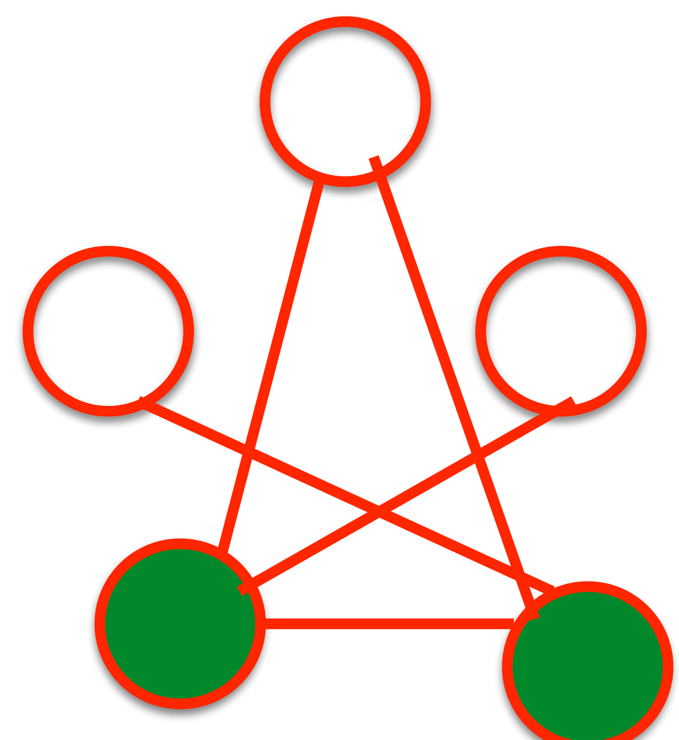
## VERTEX COVER problem

- Given a graph $G$, is there a subset of $k$ edges such that all edges have at least one endpoint in that set?

## INDEPENDENT SET reduction

- For any subset of vertices $S$, $S$ is an independent set if and only if $V - S$ is a vertex cover.
- $G$ has an independent set of size $k$ if and only if $G$ has a vertex cover of size $n - k$.





# VERTEX COVER $\leq_P$ SET COVER

## SET COVER problem

- Given sets $S_1, S_2, \ldots, S_n$, what is the smallest subset of those sets covering all of the elements of $S_1 \cup S_2 \cup \ldots S_n$?

## VERTEX COVER reduction

- Universe to cover = edges
- Sets are edges adjacent to each vertex.