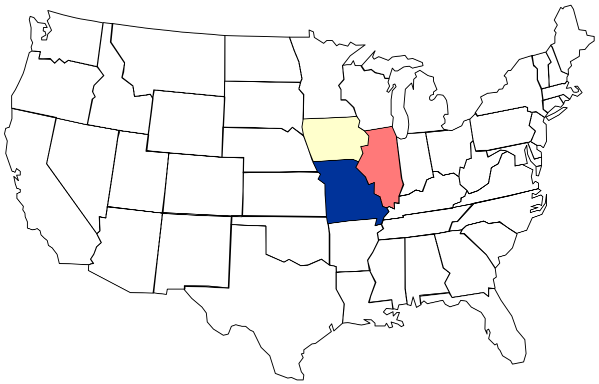
# Lecture T6: NP-Completeness



Can you color each of the 48 states red, white, or blue so that no two adjacent states have the same color?

## Overview

**Lecture T4:**
- What is an algorithm?
  - Turing machine
- Which problems can be solved on a computer?
  - not the halting problem

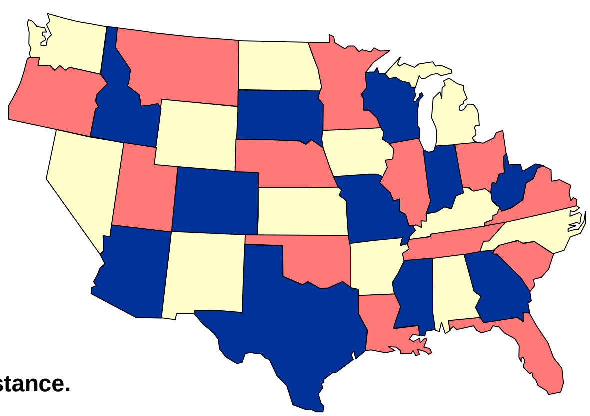
**Lecture T5:**
- Which algorithms will be useful in practice?
  - polynomial vs. exponential algorithms

**This lecture:**
- Which problems can be solved on a computer in a reasonable amount of time?
  - probably not 3-COLOR or TSP



## 3 Colorability

**3-COLOR.**
- Given a planar map, can it be colored using 3 colors so that no adjacent regions have the same color?

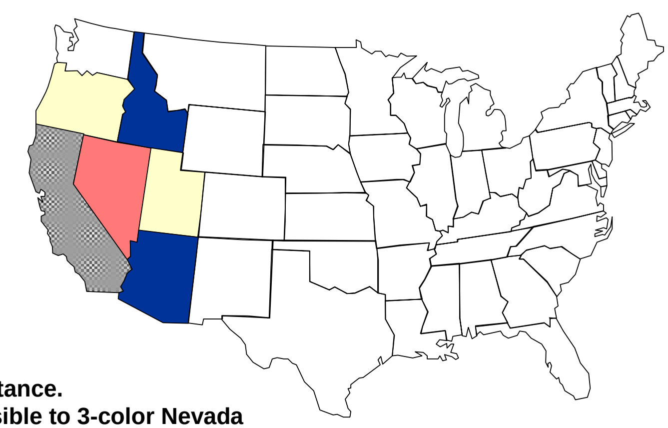


YES instance.



## 3 Colorability

**3-COLOR.**
- Given a planar map, can it be colored using 3 colors so that no adjacent regions have the same color?

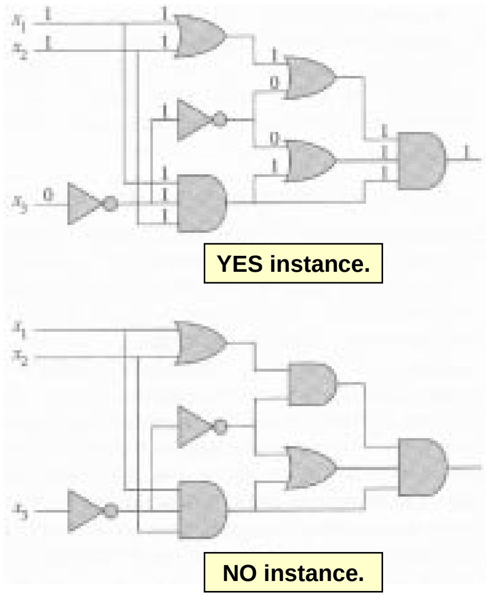


NO instance.
Impossible to 3-color Nevada and bordering states.

# Some Hard Problems

**CIRCUIT-SAT**

- Is there a way to assign inputs to a given Boolean circuit that makes it true?






# Some Hard Problems

**FACTOR**

- Given two positive integers x and U, is there a nontrivial factor of x that is less than U?
- Factoring is at the heart of RSA encryption.

**Input:** x = 23,536,481,273, U = 110,000

**Yes,** since x = 224,737 * 104,729.



# Properties of Algorithms

**A given problem can be solved by many different algorithms (TM's).**

- Which ones are useful in practice?

**A working definition: (Jack Edmonds, 1962)**

- Efficient: polynomial time for ALL inputs.
  - mergesort requires $N \log_2 N$ steps
- Inefficient: "exponential time" for SOME inputs.
  - brute force TSP takes $N! > 2^N$ steps

**Robust definition has led to explosion of useful algorithms for wide spectrum of problems.**



# Exponential Growth

**Exponential growth dwarfs technological change.**

- Suppose each electron in the universe had power of today's supercomputers.
- And each works for the life of the universe in an effort to solve TSP problem using N! algorithm from Lecture P6.

*Some Numbers*

| quantity | number |
|---|---|
| Home PC instructions/second | $10^9$ |
| Supercomputer instructions per second | $10^{12}$ |
| Seconds per year | $10^9$ |
| Age of universe in years (estimated) | $10^{13}$ |
| Electrons in universe (estimated) | $10^{79}$ |

- Will not succeed for 1,000 city TSP!

$1000! \gg 10^{1000} \gg 10^{79} * 10^{13} * 10^9 * 10^{12}$

# Properties of Problems

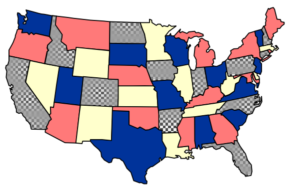
**Which ALGORITHMS will be useful in practice?**

- Efficient: polynomial time for ALL inputs.
- Inefficient: "exponential time" for SOME inputs.

**Which PROBLEMS will we be able to solve in practice?**

- Those with efficient algorithms.
- How can I tell if I am trying to solve such a problem?
  - 2-COLOR: yes
  - 3-COLOR: probably no
  - 4-COLOR: yes

Theorem (Appel-Haken, 1976).
Every planar map is 4 colorable.





# P

**Definition of P:**

- Set of all decision problems solvable in polynomial time on a deterministic Turing machine.

**Examples:**

- MULTIPLE: Is the integer y a multiple of x?
  - YES: (x, y) = (17, 51).
- RELPRIME: Are the integers x and y relatively prime?
  - YES: (x, y) = (34, 39).
- MEDIAN: Given integers $x_1, \ldots, x_n$, is the median value < M?
  - YES: $(M, x_1, x_2, x_3, x_4, x_5) = (17, 2, 5, 17, 22, 104)$

**Definition important because of Strong Church-Turing thesis.**



# Strong Church-Turing Thesis

**Strong Church-Turing thesis:**

- P is the set of all decision problems solvable in polynomial time on REAL computers.

**Evidence supporting thesis:**

- True for all physical computers.
  - can create deterministic TM that efficiently simulates TOY machine (and vice versa)
  - can create deterministic TM that efficiently simulates any physical machine (and vice versa)
- Possible exception?
  - quantum computers – no conventional gates



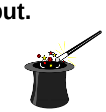
# NP

**Definition of NP:**

- Set of all decision problems solvable in polynomial time on a NONDETERMINISTIC Turing machine.
- Definition important because it links many fundamental problems.

**Useful alternate definition:**

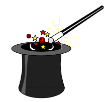
- Set of all decision problems with efficient verification algorithms.
  - efficient = polynomial number of steps on deterministic TM
- Verifier: algorithm for decision problem with extra input.



# Verifiers and Certificates

**COMPOSITE: Given integer x, is x composite?**

- YES instance: x = 23,536,481,273.
  - a corresponding certificate: c = 104,729 (a factor)
  - every YES instance has such a certificate



**Input x:** 23,536,481,273

**Certificate c:** 104,729

**Verifier:** Is x a multiple of c?

YES → x is a YES instance

NO → no conclusion



# Verifiers and Certificates

**COMPOSITE: Given integer x, is x composite?**

- YES instance: x = 23,536,481,273.
  - a corresponding certificate: c = 104,729 (a factor)
  - every YES instance has such a certificate
- NO instance: x = 23,536,481,277.
  - no NO instance has a valid certificate
- Conclusion: COMPOSITE is in NP.

**Input x:** 23,536,481,277

**Certificate c:** ??????

**Verifier:** Is x a multiple of c?

YES → x is a YES instance

NO → no conclusion



# Verifiers and Certificates

**PRIME: Given integer x, is x prime?**

- YES instance: x = 23,536,481,277.
  - not at all obvious what makes a good certificate
  - using deep facts from number theory, every YES instance has a certificate of primality
- Fact: PRIME is in NP.

**Input x:** 23,536,481,277

**Certificate c:** ??????

**Verifier:** ??????

YES → x is a YES instance

NO → no conclusion

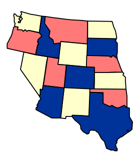


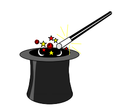
# Verifiers and Certificates

**3-COLOR: Given planar map, can it be colored with 3 colors?**

**Input x:**

**Certificate c:**





**Verifier:**
1. Check that x and c describe same map.
2. Count number of distinct colors in c.
3. Check all pairs of adjacent states.

YES → x is a YES instance

NO → no conclusion

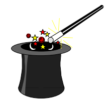
**3-COLOR is in NP.**

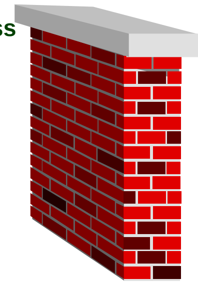
# NP

NP = set of decision problems with efficient verification algorithms.

Why doesn't this imply that all problems in NP can be solved efficiently?

- BIG PROBLEM: need to know certificate ahead of time.
  - real computers can simulate by guessing all possible certificates and verifying
  - naïve simulation takes exponential time unless you get "lucky"





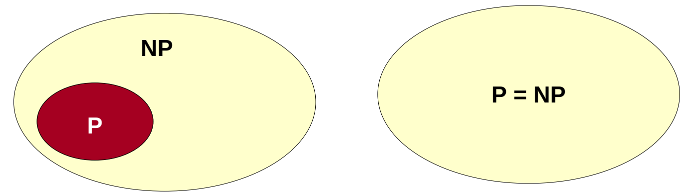


# The Main Question

Does P = NP? (Edmonds, 1962)

- Is the original DECISION problem as easy as VERIFICATION?

Most important open problem in theoretical computer science. Also ranked #3 in all of mathematics. (Smale, 1999)




If $P \neq NP$ If $P = NP$



# The Main Question

Does P = NP?

- Is the original DECISION problem as easy as VERIFICATION?

If yes, then:

- Efficient algorithms for 3-COLOR, TSP, and factoring.
- Cryptography is impossible (except for one-time pads) on conventional machines.
- Modern banking system will collapse.
- Harmonial bliss.

If no, then:

- Can't hope to write efficient algorithm for TSP.
  - see NP-completeness
- But maybe efficient algorithm still exists for factoring???



# The Main Question

Does P = NP?

- Is the original DECISION problem as easy as VERIFICATION?

Probably no, since:

- Thousands of researchers have spent four decades in search of polynomial algorithms for many fundamental NP problems without success.
- Consensus opinion: $P \neq NP$.

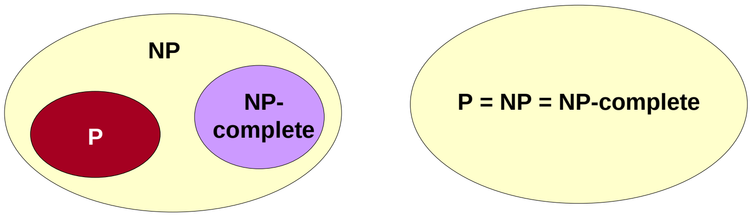
But maybe yes, since:

- No success in proving $P \neq NP$ either.

# NP-Complete

**Definition of NP-complete:**

- A problem with the property that if it can be solved efficiently, then it can be used as a subroutine to solve any other problem in NP efficiently.
- "Hardest computational problems" in NP.




If $P \neq NP$

If $P = NP$



# NP-Complete

**Definition of NP-complete:**

- A problem with the property that if it can be solved efficiently, then it can be used as a subroutine to solve any other problem in NP efficiently.

**Links together a huge and diverse number of fundamental problems:**

- TSP, 3-COLOR, SCHEDULE, SAT, CLIQUE, thousands more.
- Given an efficient algorithm for 3-COLOR, can efficiently solve TSP, SCHEDULE, SAT, CLIQUE, FACTOR, etc.
- Can implement any program in 3-COLOR.

**Note: FACTOR is in NP but not known to be NP-complete.**

**Notorious complexity class.**

- Only exponential algorithms known for these problems.
- Called intractable - unlikely that they can be solved given limited computing resources.



# Reduction

**Reduction is a general technique for showing that one problem is harder (easier) than another.**

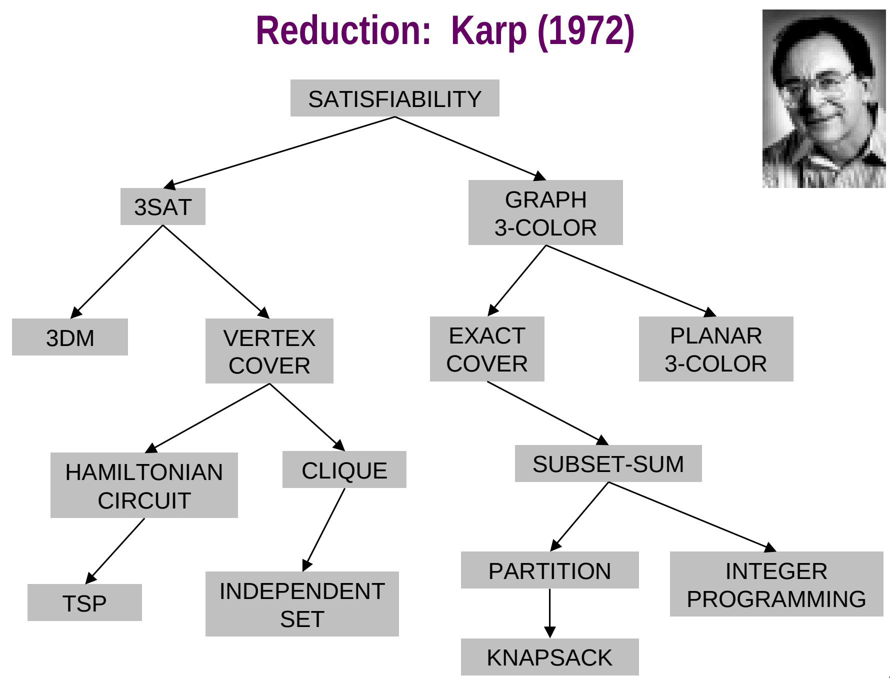
- For problems A and B, we can often show: if A can be solved efficiently, then so can B.
- In this case, we say B reduces to A. (B is "easier" than A).

**Warmup: PRIMALITY reduces to FACTOR.**

- Given any instance of PRIMALITY (i.e., positive integer x), we can determine the yes-no answer by using X = L = p as input to FACTOR and returning opposite answer.
  - original instance: Is p = 23,536,481,273 prime?
  - transformed instance: Does X = 23,536,481,273 have a nontrivial factor less than L = 23,536,481,273?
  - if answer to transformed instance is no, then answer to original instance is yes
  - if answer to transformed instances is yes, then answer to original instance is no



# Reduction: Karp (1972)

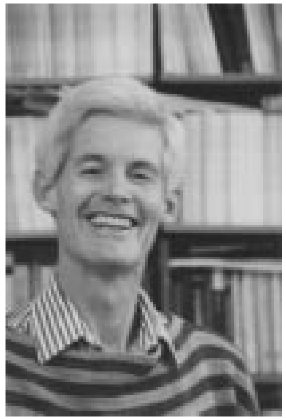


# The "World's First" NP-Complete Problem

SAT is NP-complete. (Cook-Levin, 1960's)

Idea of proof:

- By definition, nondeterministic TM can solve problem in NP in polynomial time.
- Polynomial-size Boolean formula can describe (nondeterministic) TM.
- Given any problem in NP, establish a correspondence with some instance of SAT.
- SAT solution gives simulation of TM solving the corresponding problem.
- IF SAT can be solved in polynomial time, then so can any problem in NP (e.g., TSP).



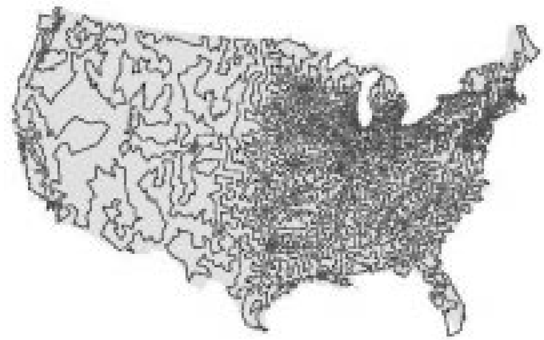
Stephen Cook



# Coping With NP-Completeness

Hope that worst case doesn't occur.

- Complexity theory deals with worst case behavior. The instance(s) you want to solve may be "easy."
  - TSP where all points are on a line or circle
  - 13,509 US city TSP problem solved (Cook et. al., 1998)



Bill Cook



# Coping With NP-Completeness

Hope that worst case doesn't occur.

Change the problem.

- Develop a heuristic, and hope it produces a good solution.
  - TSP assignment.
- Design an approximation algorithm: algorithm that is guaranteed to find a high-quality solution in polynomial time.
  - active area of research, but not always possible
  - Euclidean TSP tour within 1% of optimal (Arora, 1997)

Sanjeev Arora



# Coping With NP-Completeness

Hope that worst case doesn't occur.

Change the problem.

Exploit intractability.

Keep trying to prove P = NP.

# Summary

**Many fundamental problems are NP-complete.**

- **TSP, SAT, 3-COLOR.**

**Theory says we probably won't be able to design efficient algorithms for NP-complete problems.**

- **You will likely run into these problems in your scientific life.**
- **If you know about NP-completeness, you can identify them and avoid wasting time.**

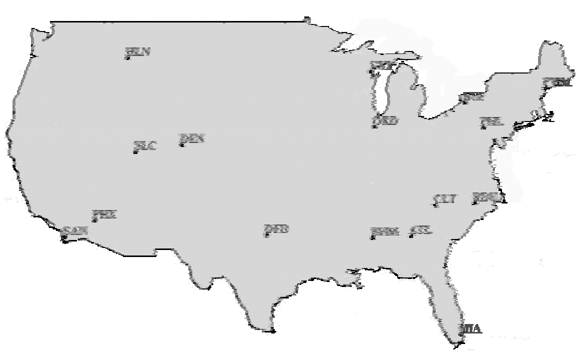


# Lecture T6: Extra Slides

# Some Hard Problems

**TSP**

- **A travelling salesperson needs to visit N cities. Is there a route of length at most D?**



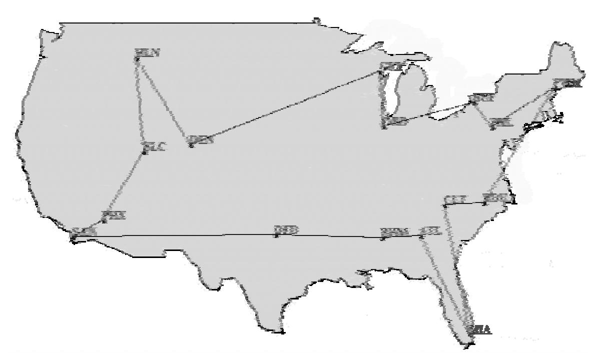
Is there a tour of length at most 1570?



# Some Hard Problems

**TSP**

- **A travelling salesperson needs to visit N cities. Is there a route of length at most D?**

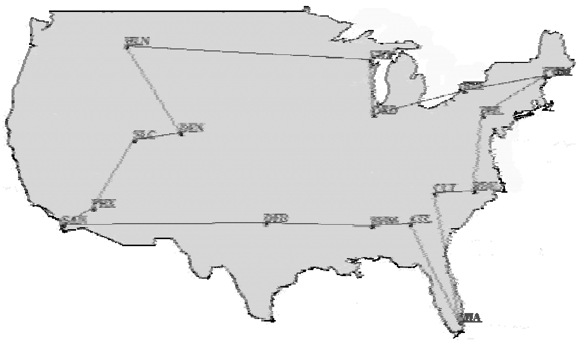


Is there a tour of length at most 1570? Blue tour = 1581.

# Some Hard Problems

**TSP**

- **A travelling salesperson needs to visit N cities. Is there a route of length at most D?**



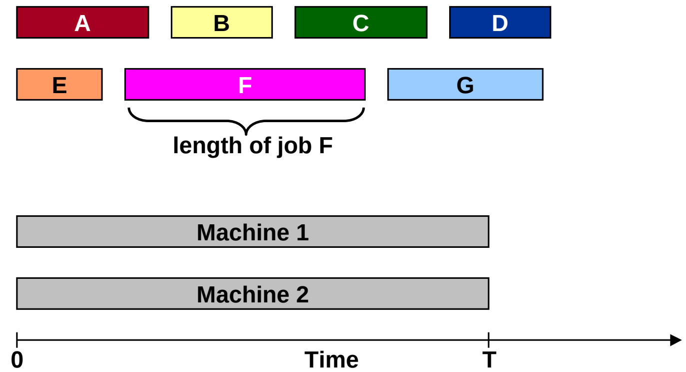
Is there a tour of length at most 1570? Yes, red tour = 1565.



# Some Hard Problems

**SCHEDULE**

- **A set of jobs of varying length need to be processed on two identical machines before a certain deadline T. Can the jobs be arranged so that the deadline is met?**

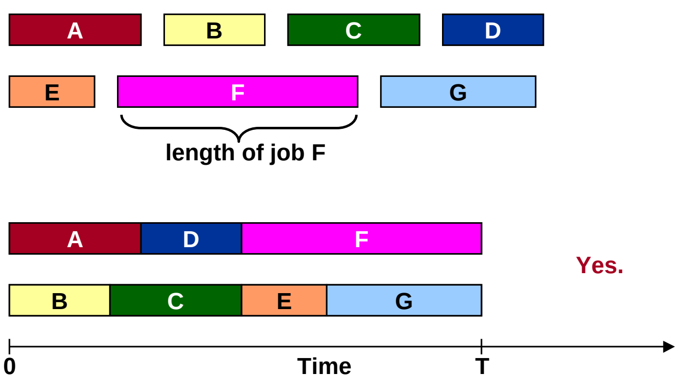





# Some Hard Problems

**SCHEDULE**

- **A set of jobs of varying length need to be processed on two identical machines before a certain deadline T. Can the jobs be arranged so that the deadline is met?**




**Yes.**



# Some Hard Problems

**CLIQUE**

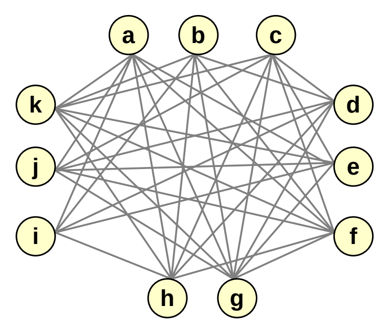
- **Given N people and their pairwise relationships. Is there a group of S people such that every pair in the group knows each other.**

**People:** **a, b, c, d, e, . . ., k**

**Friendships:** **(a, e), (a, f), (a, g), . . ., (h, k)**

**Clique size:** **S = 4?**

**Friendship Graph**



## Some Hard Problems

**CLIQUE**

- Given N people and their pairwise relationships. Is there a group of S people such that every pair in the group knows each other.

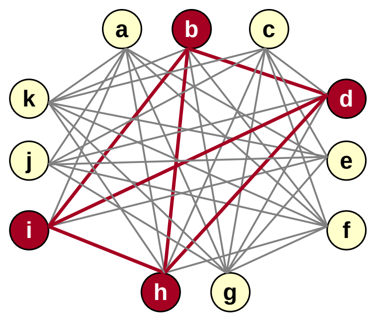
**People:** a, b, c, d, e, . . ., k

**Friendships:** (a, e), (a, f), (a, g), . . ., (h, k)

**Clique size:** S = 4?

**Yes - {b, d, i, h} is a witness.**

**Friendship Graph**






## Some Hard Problems

**SAT**

- Is there a way to assign truth values to a given Boolean formula that makes it true?

**Boolean formula:** (x' + y + z) (x + y' + z) (y + z) (x' + y' + z')

**Yes, x = true, y = true, z = false is a witness.**



## Reduction

**Reduction is a general technique for showing that one problem is harder (easier) than another.**

- For problems A and B, we can often show: if A can be solved efficiently, then so can B.
- In this case, we say B reduces to A. (B is "easier" than A).

**SAT reduces to CLIQUE**

- Given any input to SAT, we create a corresponding input to CLIQUE that will help us solve the original SAT problem.
- Specifically, for a SAT formula with K clauses, we construct a CLIQUE input that has a clique of size K if and only if the original Boolean formula is satisfiable.
- If we had an efficient algorithm for CLIQUE, we could apply our transformation, solve the associated CLIQUE problem, and obtain the yes-no answer for the original SAT problem.



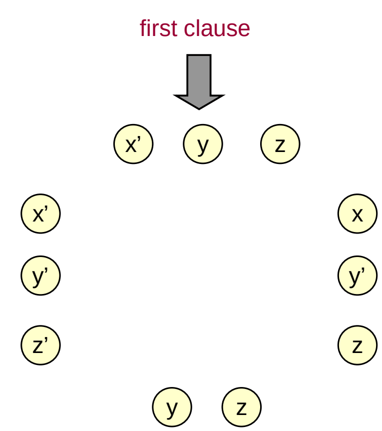
## SAT reduces to CLIQUE

**SAT reduces to CLIQUE**

- Associate a person to each variable occurrence in each clause.

Boolean formula:

(x' + y + z) (x + y' + z) (y + z) (x' + y' + z')



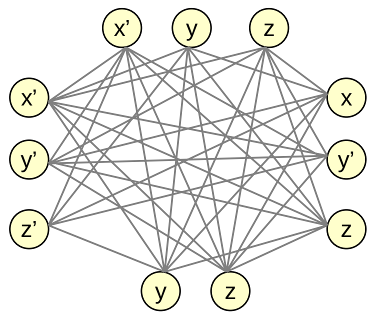
# SAT reduces to CLIQUE

**SAT reduces to CLIQUE**

- **Associate a person to each variable occurrence in each clause.**
- **Two people know each other except if:**
  - **they come from the same clause**
  - **they represent t and t' for some variable t**

Boolean formula:

(x' + y + z) (x + y' + z) (y + z) (x' + y' + z')





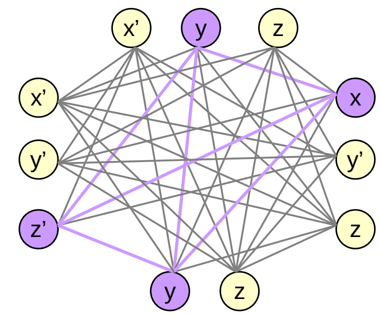

# SAT reduces to CLIQUE

**SAT reduces to CLIQUE**

- **Associate a person to each variable occurrence in each clause.**
- **Two people know each other except if:**
  - **they come from the same clause**
  - **they represent t and t' for some variable t**
- **Clique of size 4 ⇒ satisfiable assignment.**
  - **set variable in clique to true**
  - **(x, y, z) = (true, true, false)**

Boolean formula:

(x' + y + z) (x + y' + z) (y + z) (x' + y' + z')





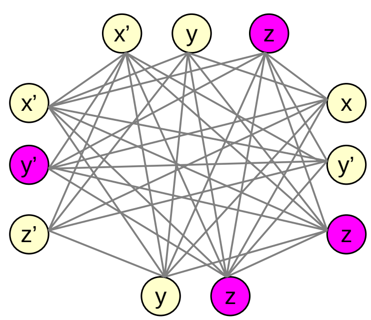

# SAT reduces to CLIQUE

**SAT reduces to CLIQUE**

- **Associate a person to each variable occurrence in each clause.**
- **Two people know each other except if:**
  - **they come from the same clause**
  - **they represent t and t' for some variable t**
- **Clique of size 4 ⇒ satisfiable assignment.**
- **Satisfiable assignment ⇒ clique of size 4**
  - **(x, y, z) = (false, false, true)**
  - **choose one true literal from each clause**

Boolean formula:

(x' + y + z) (x + y' + z) (y + z) (x' + y' + z')






# CLIQUE is NP-Complete

**CLIQUE is NP-complete.**

- **CLIQUE is in NP.**
- **SAT is NP-complete.**
- **SAT reduces to CLIQUE.**

**Thousands of problems shown to be NP-complete in this way.**