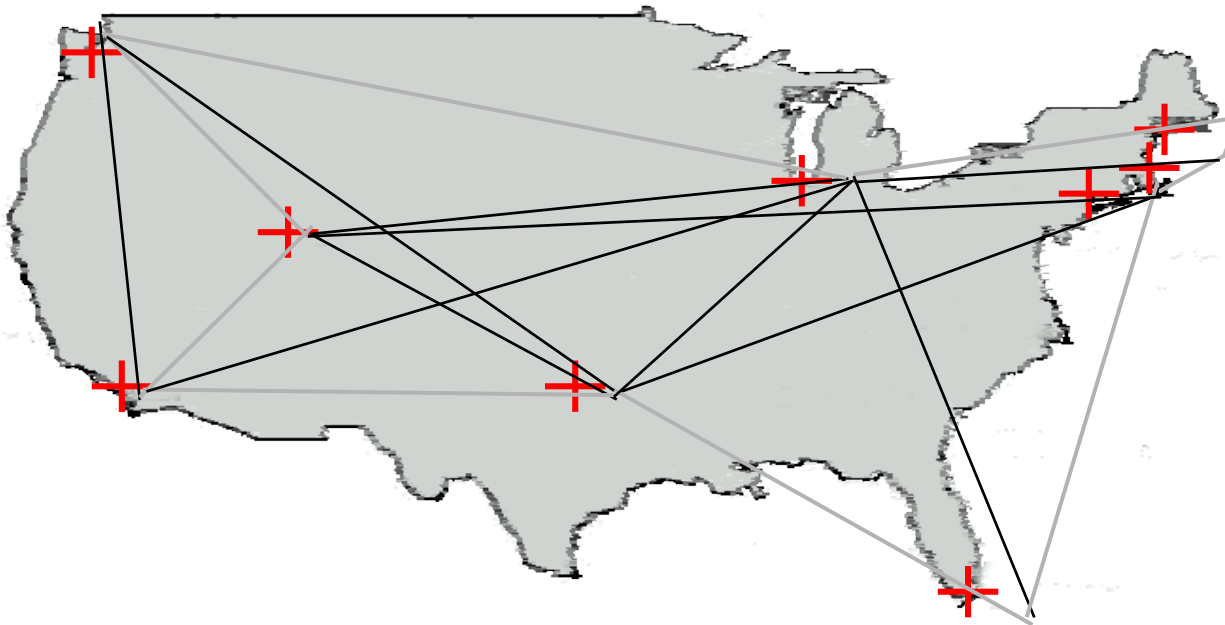


# Intractability

- Tractable and intractable problems
  - What is a "reasonable" running time?
  - NP problems, examples
  - NP-complete problems and polynomial reducability
- There are many practically important problems that have not yielded algorithms with sub-exponential worst case running time even with years of effort.

# Traveling Salesman Problem

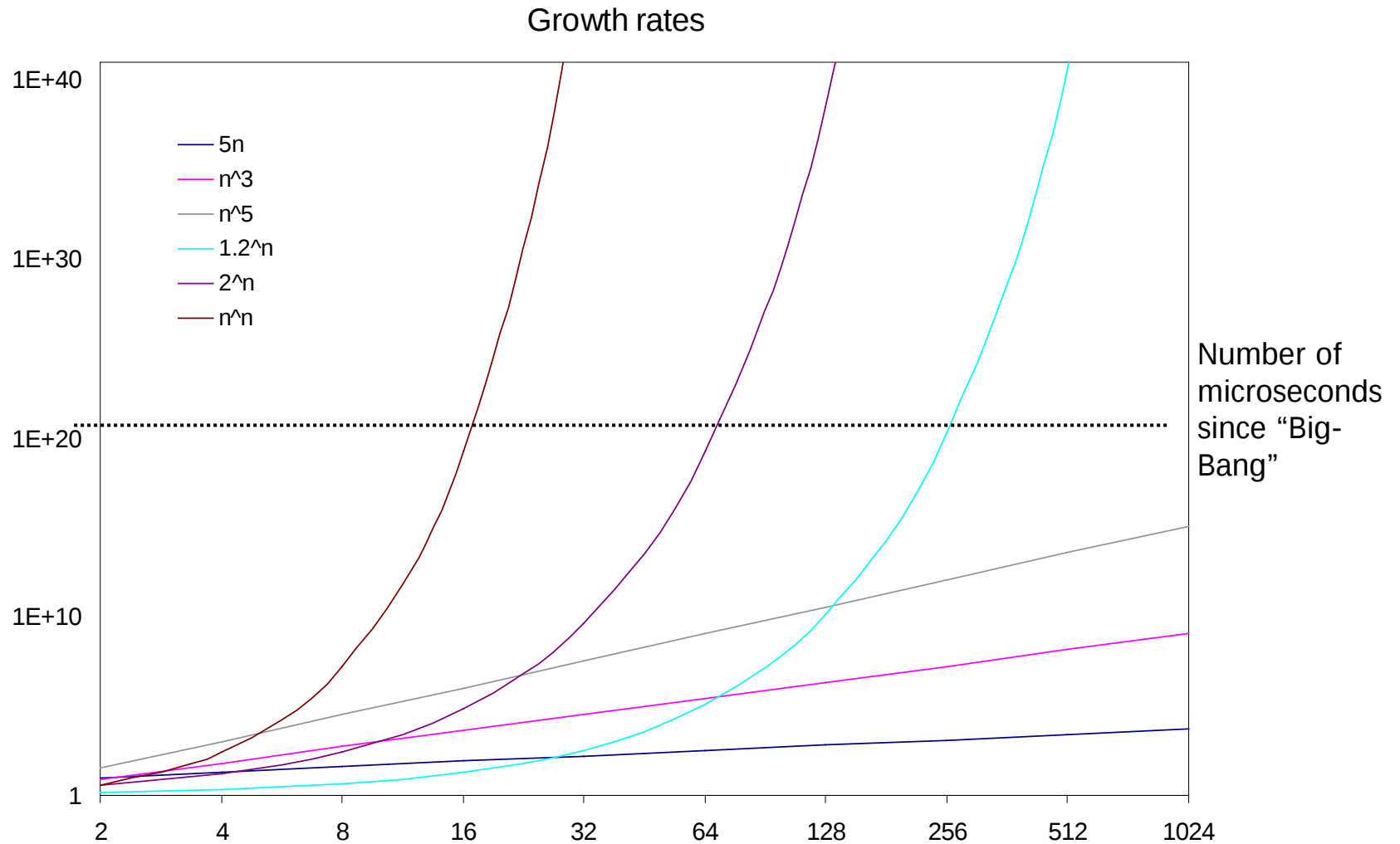
- A traveling salesperson needs to visit  $n$  cities
- Is there a route of at most  $d$  length? (decision problem)
  - Optimization-version asks to find a shortest cycle visiting all vertices once in a weighted graph



# TSP Algorithms

- Naive solutions take  $n!$  time in worst-case, where  $n$  is the number of edges of the graph
- No polynomial-time algorithms are known
  - TSP is an NP-complete problem
- Longest Path problem between A and B in a weighted graph is also NP-complete
  - Remember the running time for the shortest path problem

# Reasonable vs. Unreasonable

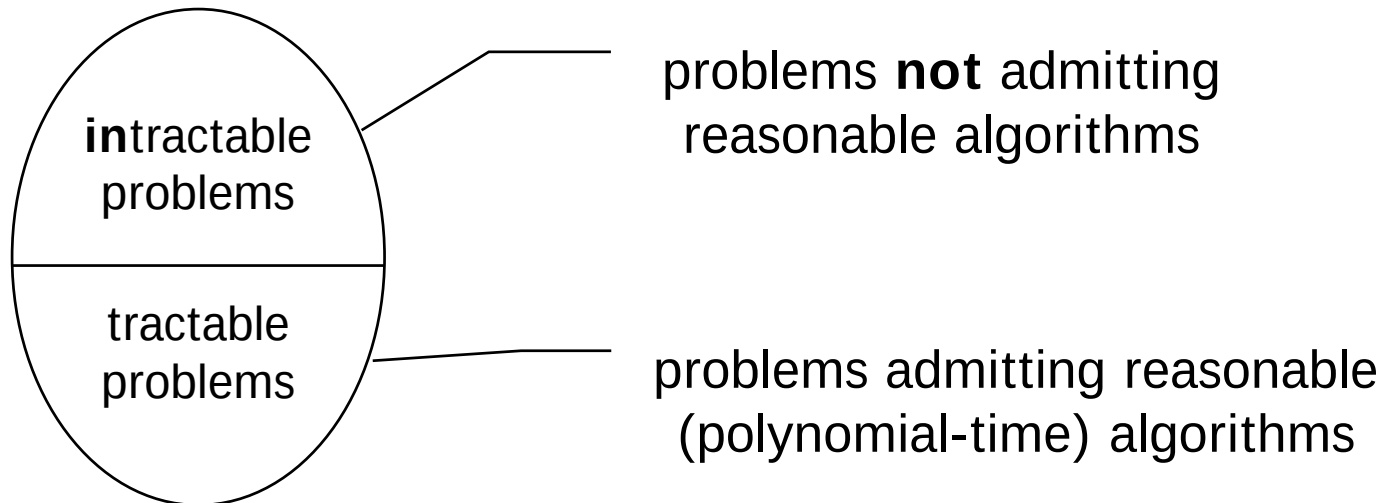


# Reasonable vs. Unreasonable

|             |  | function/<br>$n$ | 10                 | 20                    | 50                                    | 100                                    | 300                                    |
|-------------|--|------------------|--------------------|-----------------------|---------------------------------------|----------------------------------------|----------------------------------------|
| Polynomial  |  | $n^2$            | 1/10,000<br>second | 1/2,500<br>second     | 1/400<br>second                       | 1/100<br>second                        | 9/100<br>second                        |
|             |  | $n^5$            | 1/10<br>second     | 3.2<br>seconds        | 5.2<br>minutes                        | 2.8<br>hours                           | 28.1<br>days                           |
| Exponential |  | $2^n$            | 1/1000<br>second   | 1<br>second           | 35.7<br>years                         | 400 trillion<br>centuries              | a 75 digit-<br>number of<br>centuries  |
|             |  | $n^n$            | 2.8<br>hours       | 3.3 trillion<br>years | a 70 digit-<br>number of<br>centuries | a 185 digit-<br>number of<br>centuries | a 728 digit-<br>number of<br>centuries |

## Reasonable vs. Unreasonable

- "Good", reasonable algorithms
  - algorithms bound by a polynomial function  $n^k$
  - **Tractable problems**
- "Bad", unreasonable algorithms
  - algorithms whose running time is above  $n^k$
  - **Intractable problems**

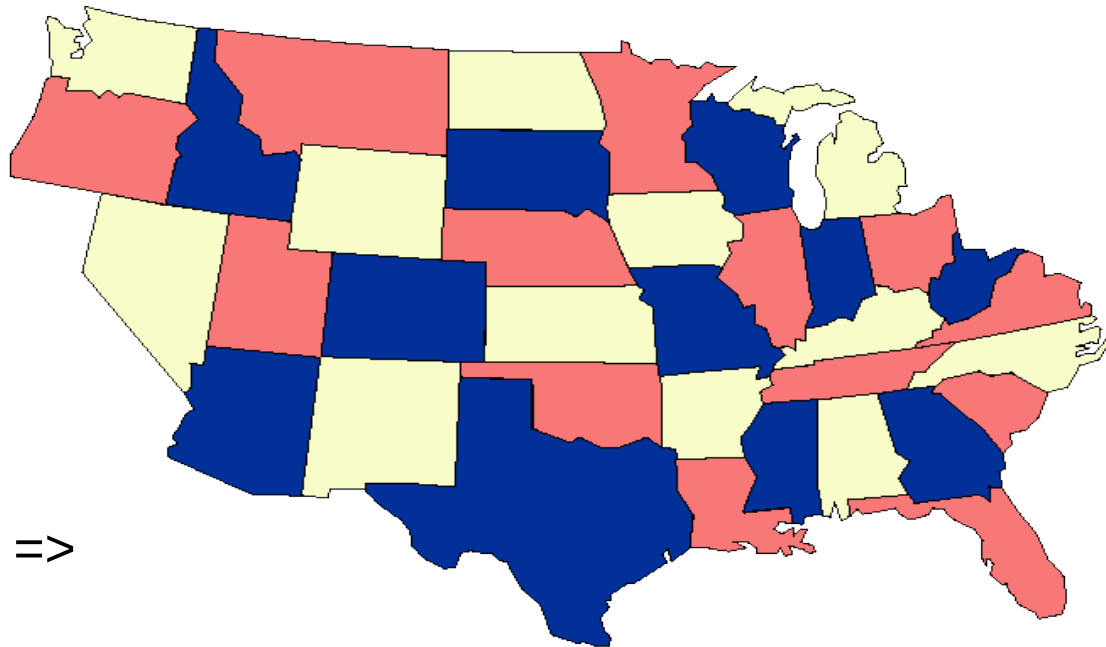


## Counterpoints?

- Computers become faster every day
  - insignificant (a constant) compared to exp. running time
- Maybe the TSP is just one specific problem, we could simply ignore?
  - the TSP falls into a category of problems called NPC (NP complete) problems (~1000 problems)
  - all admit **unreasonable** solutions
  - **not known** to admit **reasonable** ones...

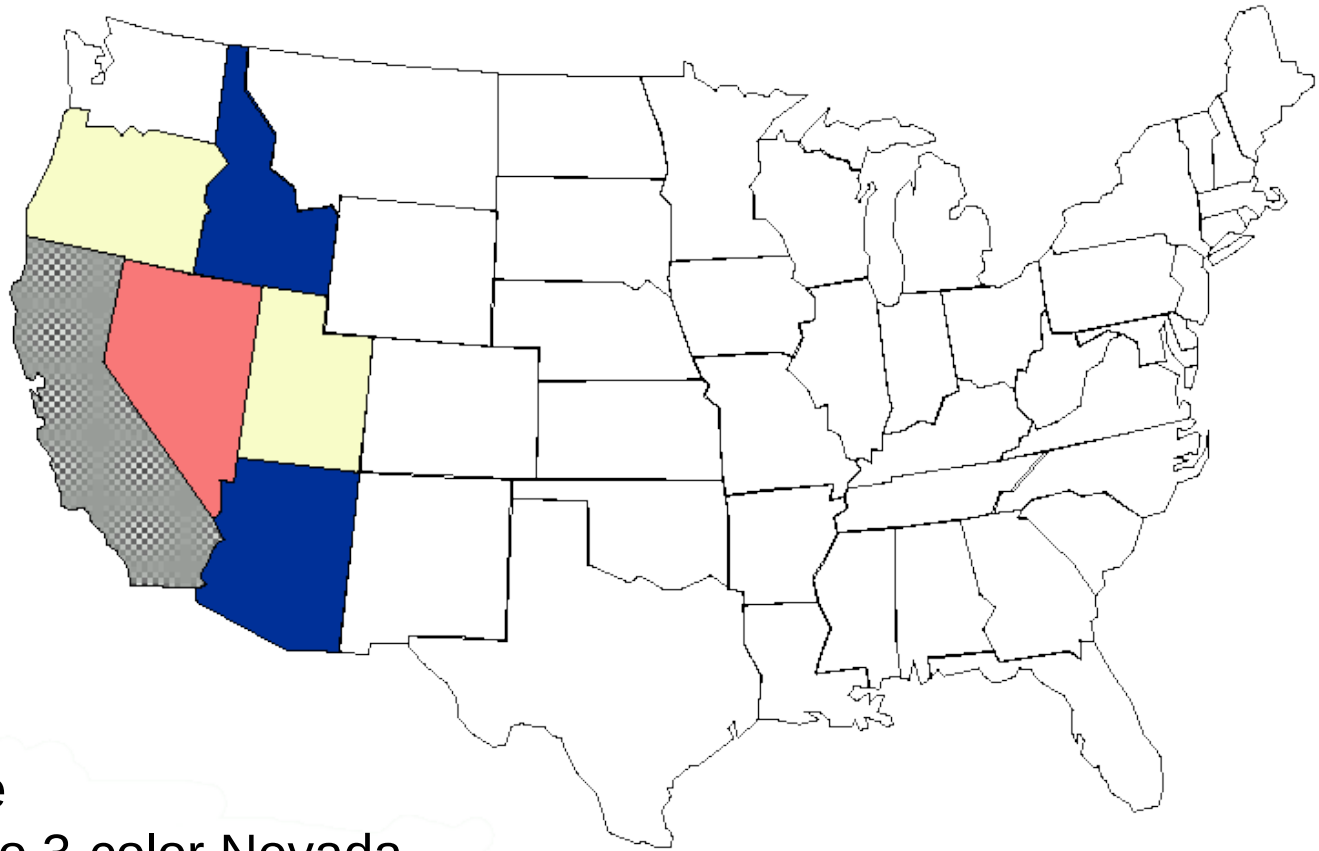
# Coloring Problem (COLOR)

- 3-color
  - given a planar map, can it be colored using 3 colors so that no adjacent regions have the same color



Find an error! =>

## Coloring Problem (2)



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

## Coloring Problem (3)

- Any map can be **4-colored**
- Maps that contain no points that are the junctions of an odd number of states can be **2-colored**
- No polynomial algorithms are known to determine whether a map can be **3-colored** – it's an NP-complete problem

## Determining Truth (SAT)

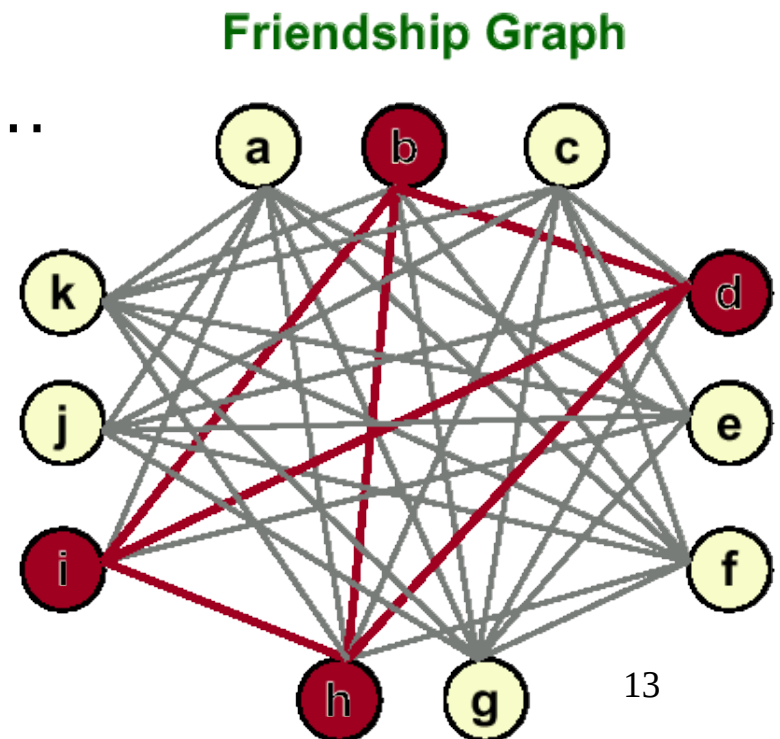
- Determine the truth or falsity of logical sentences in a simple logical formalism called **propositional calculus**
- Using the logical connectives (&-and,  $\vee$ -or,  $\sim$ -not,  $\rightarrow$ -implies) we compose expressions such as the following
$$\sim(E \rightarrow F) \ \& \ (F \vee (D \rightarrow \sim E))$$
- The algorithmic problem calls for determining the **satisfiability** of such sentences
  - e.g.,  $E = \text{true}$ ,  $D$  and  $F = \text{false}$

## Determining Truth (SAT)

- Exponential time algorithm on  $n$  = the number of distinct elementary assertions ( $\Theta(2^n)$ )
- Best known solution, problem is in NP-complete class!

# 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:  $a, b, c, \dots, k$
  - friendships:  $(a,e), (a,f), \dots$
  - clique size:  $s = 4$ ?
  - YES,  $\{b, d, i, h\}$  is a certificate!



# P

- Definition of P:
  - Set of all decision problems solvable in polynomial time on a deterministic Turing machine
- Examples:
  - SHORTEST PATH: Is the shortest path between  $u$  and  $v$  in a graph shorter than  $k$ ?
  - RELPRIME: Are the integers  $x$  and  $y$  relatively prime?
    - YES:  $(x, y) = (34, 39)$ .
  - MEDIAN: Given integers  $x_1, \dots, x_n$ , is the median value  $< M$ ?
    - YES:  $(M, x_1, x_2, x_3, x_4, x_5) = (17, 2, 5, 17, 22, 104)$

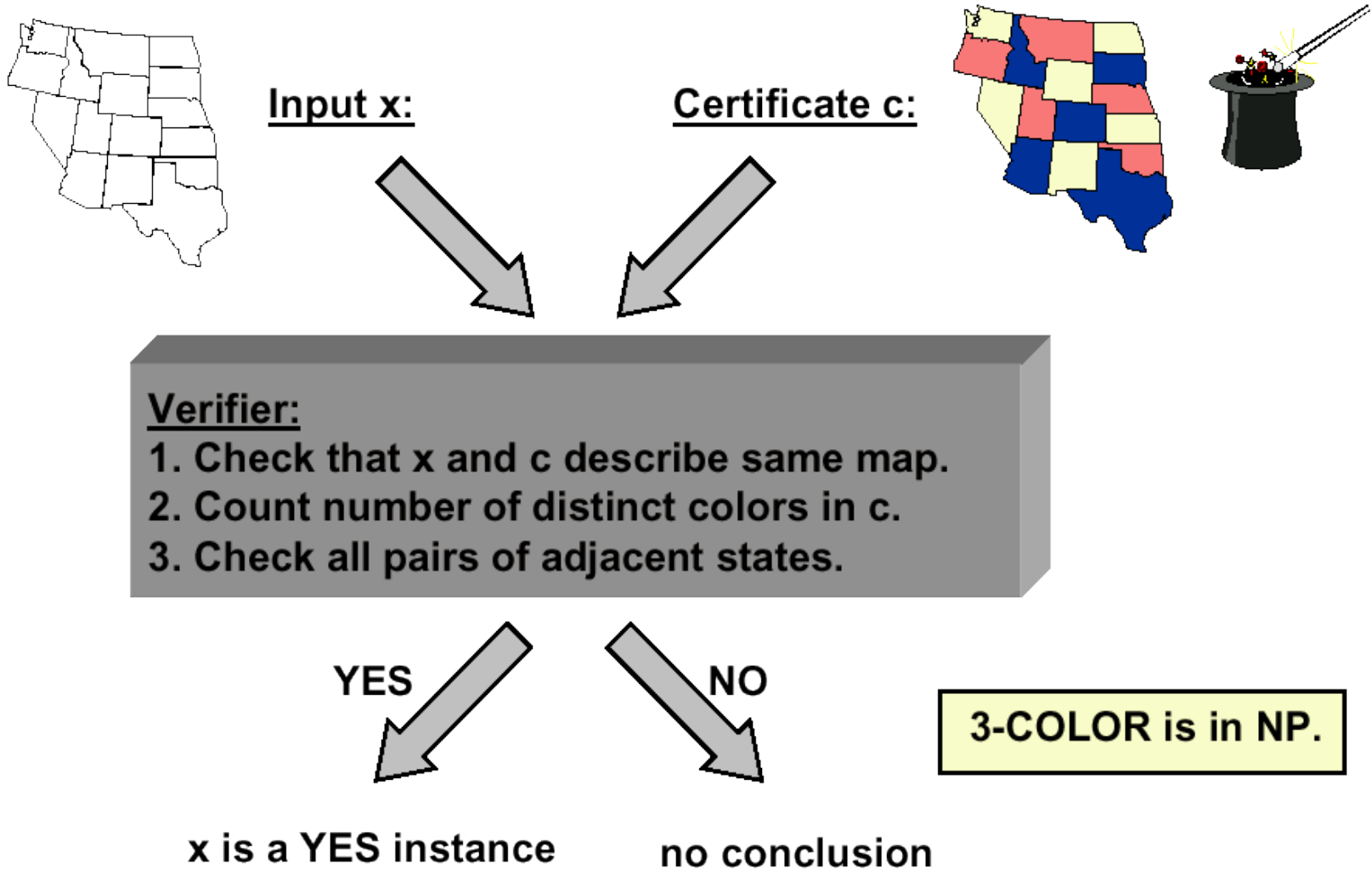
## P(2)

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

## Short Certificates

- To find a solution for an NPC problem, we seem to be required to try out exponential amounts of partial solutions
- Failing in extending a partial solution requires **backtracking**
- However, once we found a solution, convincing someone of it is easy, if we keep a proof, i.e., a **certificate**
- The problem is finding an answer (exponential), but not verifying a potential solution (polynomial)

## Short Certificates (2)



## On Magic Coins and Oracles

- Assume we use a magic coin in the backtracking algorithm
  - whenever it is possible to extend a partial solutions in  $> 1$  ways, we toss a magic coin (next city, next truth assignment, etc.)
  - the outcome of this "act" determines further actions – we use magical insight, supernatural powers!
- Such algorithms are termed "**non-deterministic**"
  - they **guess** which option is better, **rather** than employing some **deterministic procedure** to go through the alternatives

# 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 alternative 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

## NP (2)

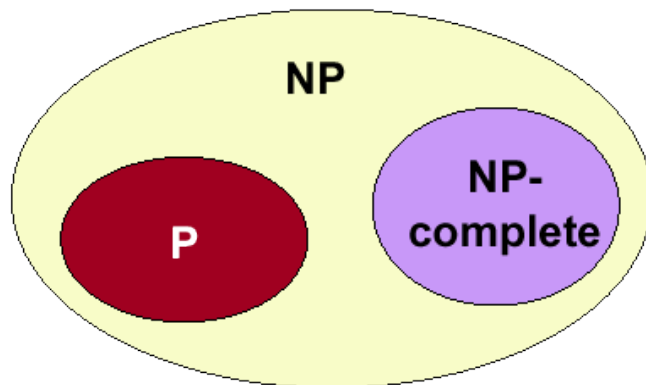
- NP = set of decision problems with efficient verification algorithms
- Why doesn't this imply that all problems in NP can be solved efficiently?
  - BIG PROBLEM: need to know certificate ahead of time
    - real computers can simulate by guessing all possible certificates and verifying
    - naïve simulation takes exponential time unless you get "lucky"

# NP-Completeness

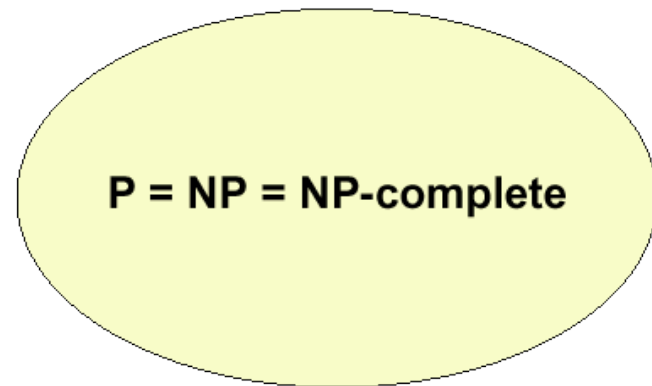
- Informal definition of NP-hard:
  - A problem with the property that if it can be solved efficiently, then it can be used as a subroutine to solve any other problem in NP efficiently
- NP-complete problems are NP problems that are NP-hard
  - "Hardest computational problems" in NP

# The Main Question

- Does  $P = NP$ ?
  - Is the original DECISION problem as easy as VERIFICATION?
- Most important open problem in theoretical computer science. Clay institute of mathematics offers one-million dollar prize!



If  $P \neq NP$



If  $P = NP$

## The Main Question (2)

- If  $P=NP$ , then:
  - Efficient algorithms for 3- COLOR, TSP, and factoring.
  - Cryptography is impossible on conventional machines
  - Modern banking systems will collapse
- If no, then:
  - Can't hope to write efficient algorithm for TSP
    - see NP- completeness
  - But maybe efficient algorithm still exists for testing the primality of a number – i.e., there are some problems that are NP, but not NP-complete

## The Main Question (3)

- Probably no, since:
  - Thousands of researchers have spent four decades in search of polynomial algorithms for many fundamental NP-complete problems without success
  - Consensus opinion:  $P \neq NP$
- But maybe yes, since:
  - No success in proving  $P \neq NP$  either

## Dealing with NP-Completeness

- Hope that a 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)
- Change the problem
  - Develop a heuristic, and hope it produces a good solution.
  - 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
- Keep trying to prove  $P = NP$ .

## The Big Picture

- It is not known whether NP problems are tractable or intractable
- But, there exist provably intractable problems
  - Even worse – there exist problems with running times far worse than exponential!
- More bad news: there are **provably noncomputable (undecidable)** problems
  - There are no (and there will not ever be!!!) algorithms to solve these problems

## Proving NP-completeness: the start...

- The World's first NP-complete problem
- SAT is NP-complete (Cook-Levin, 196x)

## Proving NP-Completeness (2)

- Each NPC problem's faith is tightly coupled to all the others (complete set of problems)
- Finding a **polynomial time algorithm for one NPC problem** would **automatically** yield a polynomial time algorithm **for all NP problems**
- Proving that one NP-complete problem has an **exponential lower bound** would **automatically** prove that **all other NP-complete** problems have exponential lower bounds

## NP-Completeness (3)

- *How can we prove such a statement?*
- Polynomial time reduction!
  - given two problems
  - it is an algorithm running in polynomial time that reduces one problem to the other such that
    - given input X to the first and asking for a yes/no answer
    - we transform X into input Y to the second problem such that its answer matches the answer of the first problem

## Reduction Example

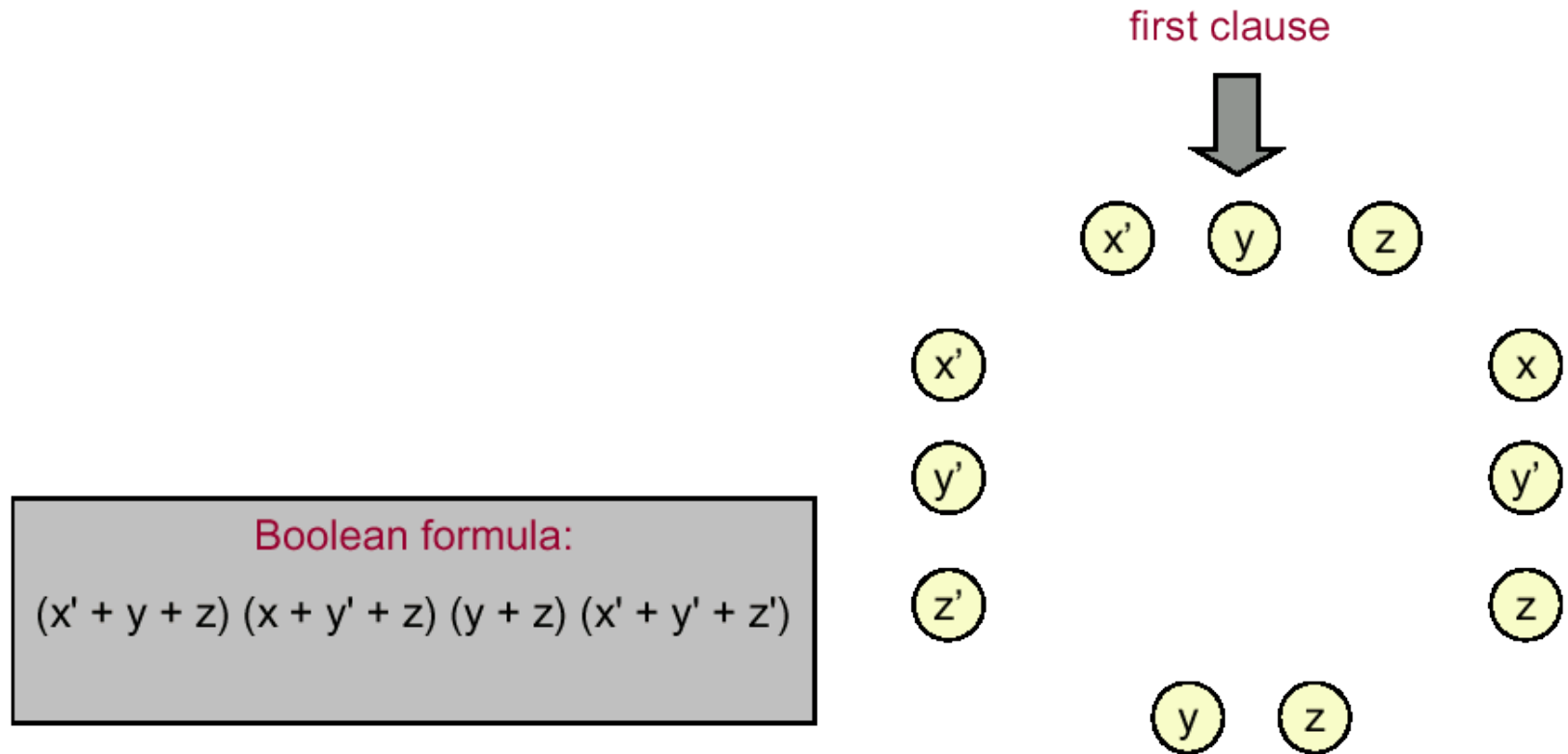
- Reduction is a general technique for showing that one problem is harder (easier) than another
  - For problems A and B, we can often show: if A can be solved efficiently, then so can B
  - In this case, we say B reduces to A (B is "easier" than A, or, B cannot be "worse" than A)

## Reduction Example (2)

- 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

## Reduction Example (3)

- SAT reduces to CLIQUE
  - Associate a person to each variable occurrence in each clause

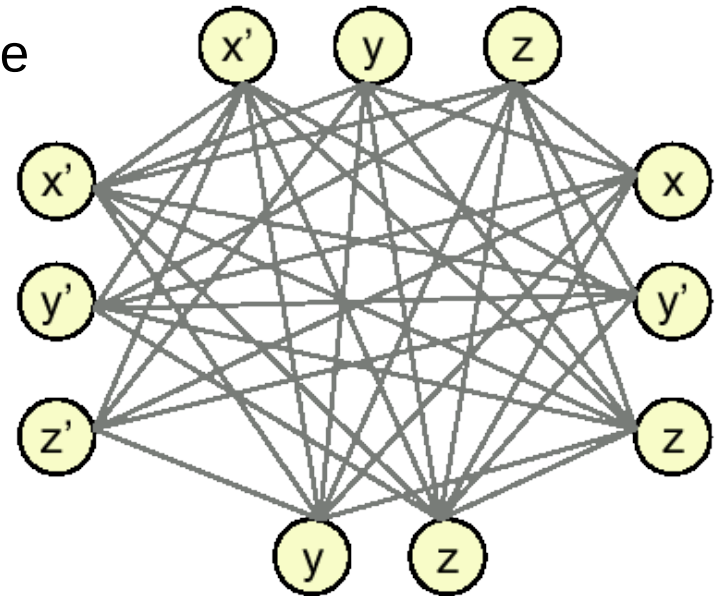


## Reduction Example (4)

- 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:

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

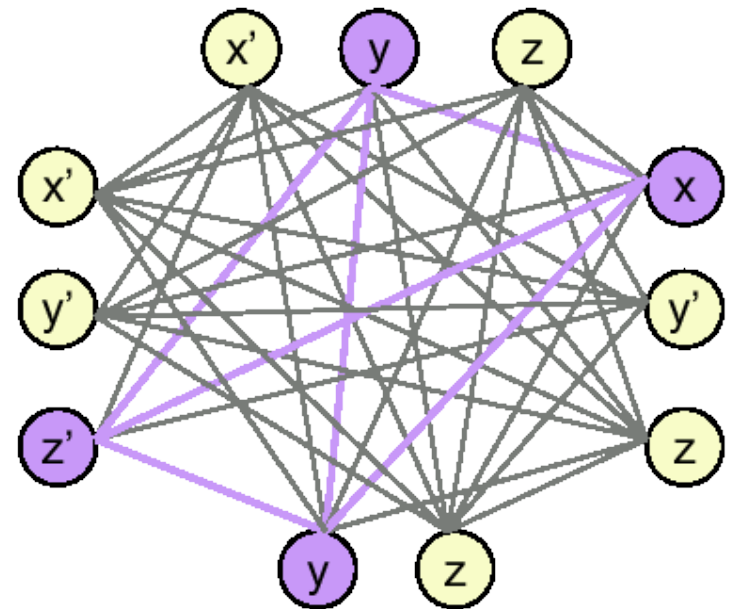


## Reduction Example (5)

- SAT reduces to CLIQUE
  - 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  $\Rightarrow$  satisfiable assignment
    - set variable in clique to "true"
    - $(x, y, z) = (\text{true}, \text{true}, \text{false})$

Boolean formula:

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

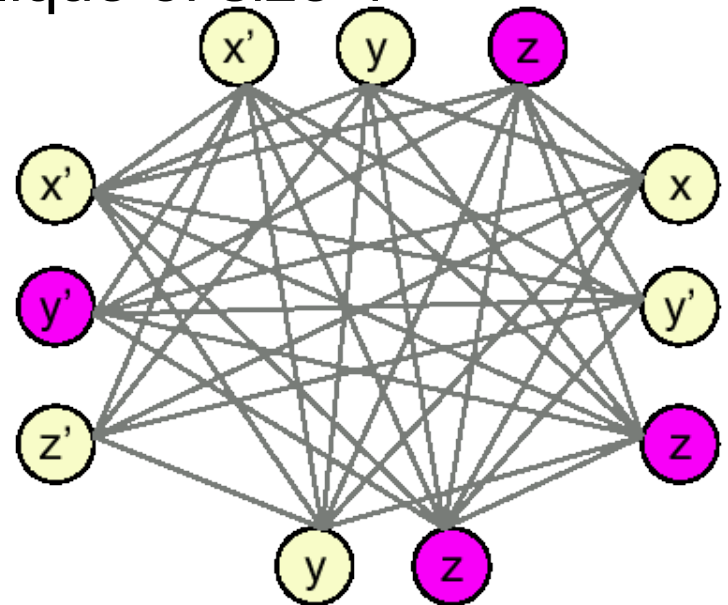


## Reduction Example (6)

- SAT reduces to CLIQUE
  - 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  $\Rightarrow$  satisfiable assignment
  - Satisfiable assignment  $\Rightarrow$  clique of size 4
    - $(x, y, z) = (\text{false}, \text{false}, \text{true})$
    - 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 in NP-complete
  - SAT reduces to CLIQUE
- Hundreds of problems can be shown to be NP-complete that way...

## Summary

- Thousands of problems have been proved to be NP-complete
  - "at least as hard as any other problem in NP"
  - If you find a polynomial time solution to any NP-complete problem,  $P=NP$
- They are believed to be intractable (i.e., no polynomial time algorithms exist)
- Since this has not been proved, it is possible that  $P=NP$ .
- In real life one looks for an approximation algorithm or a different problem formulation...