Inefficiency and Intractability

Juan Mendivelso
Profesor Asistente
Universidad Nacional de Colombia

Introduction

- There are problems for which:
  - are solved by different algorithms.
  - are solved by some algorithms but they take forever (and ever)!
  - there are no known algorithms that solve them.

Towers of Hanoi (I)

What is the time complexity of the best algorithm that solves it?

Towers of Hanoi (II)

- The time complexity of the best known algorithm is 2^N -1.
- The lower bound of the required number of moves is also 2^N -1.
- The solution is optimal!

Towers of Hanoi (III)

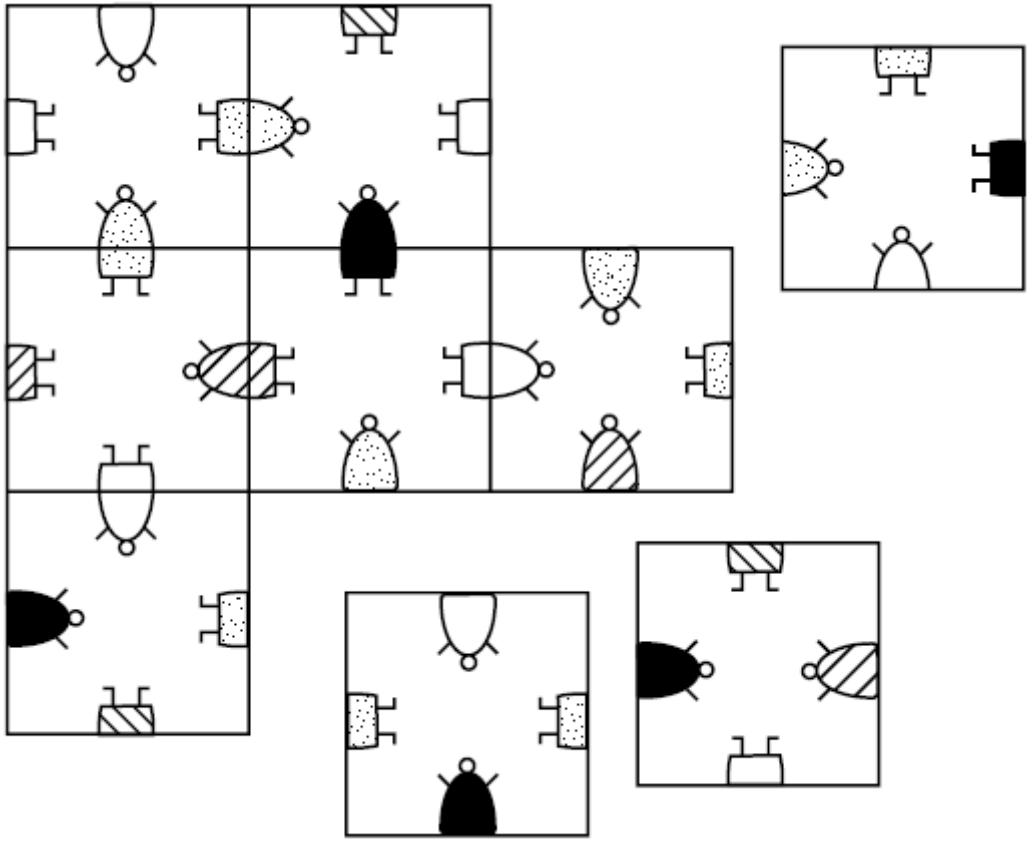
- The original problem had 64 rings. If one ring could be moved every 10 seconds, it would take over 5 trillion years to solve the problem.
- Even moving 1 million rings a second takes over 1/2 million years

Decision Problems

- Perhaps part of the long running time of an algorithm is that we are looking for a solution.
- What if we just asked, “Does a solution exist?”
- A *decision* problem is one which has a **yes** or **no** answer.

- Most problems have a decision problem version.
- For example: *What is the minimum grade I can get on the final and still get an A in the class?*
- The decision problem version is: *If I get a (some number) on the final, will I still get an A in the class?*

The Monkey Puzzle (I)



The Monkey Puzzle (II)

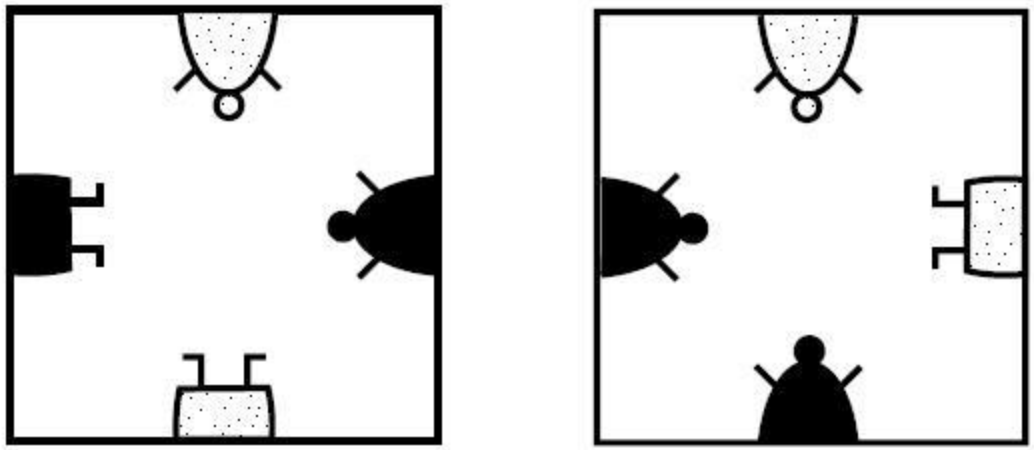
- Would you please give a naive algorithm to solve this problem?
- What is its time complexity?

The Monkey Puzzle (III)

- Given N cards, a brute-force solution takes N! steps.
- Arranging 1 million cards a second still takes over 490 billion years for 25 cards.
- For N=36, it would take far far longer than the time elapsed since the Big Bang!

The Monkey Puzzle (IV)

- Arrangements that are easier to solve:
  - All N cards are either of:



The Monkey Puzzle (V)

- The worst case is what matters.
- Better version: do not extend illegal partial arrangements.
- The performance of the most efficient algorithms are not much better that the naive one.
- Can there be an algorithm that takes reasonable time?

Different Functions of Time Complexity (I)

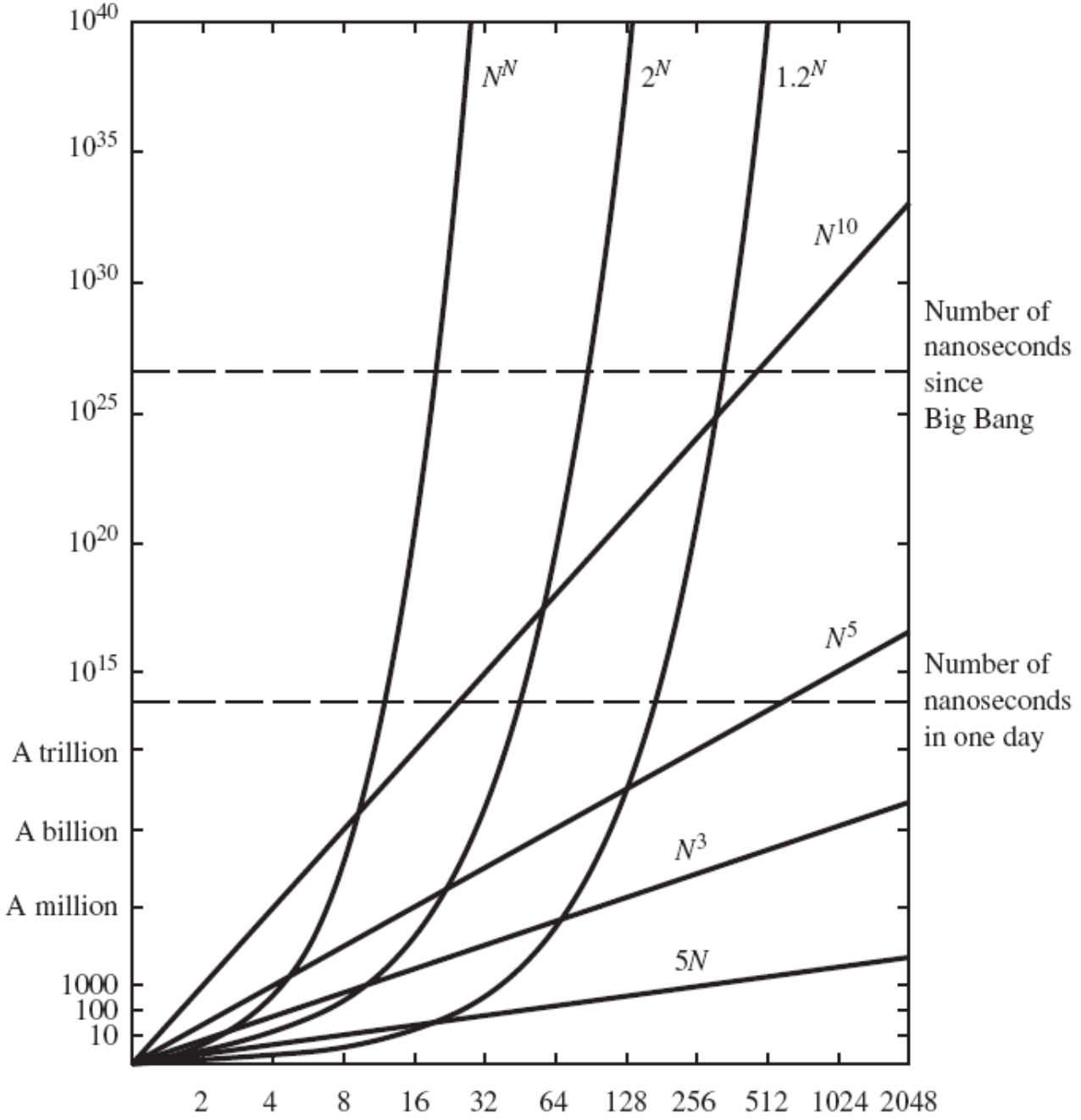
- N^K vs N!
- N! vs N^N
- N! vs 2^N
- 2^N vs N^K
- What is the order?

Different Functions of Time Complexity (II)

- Algorithms whose run time is bounded from above by a polynomial function of N are called *polynomial-time algorithms.*
- The other algorithms are called *super-polynomial-time algorithms.* We'll call them *exponential algorithms.*

Different Functions of Time Complexity (III)

	Function \ N	20	60	100	300	1000
Polynomial	$5N$	100	300	500	1500	5000
	$N \times \log_2 N$	86	354	665	2469	9966
	N^2	400	3600	10,000	90,000	1 million (7 digits)
	N^3	8000	216,000	1 million (7 digits)	27 million (8 digits)	1 billion (10 digits)
Exponential	2^N	1,048,576	a 19-digit number	a 31-digit number	a 91-digit number	a 302-digit number
	$N!$	a 19-digit number	an 82-digit number	a 161-digit number	a 623-digit number	unimaginably large
	N^N	a 27-digit number	a 107-digit number	a 201-digit number	a 744-digit number	unimaginably large


10^{40}
10^{35}
10^{30}
10^{25}
10^{20}
10^{15}
A trillion
A billion
A million
1000
100
10
N^N
2^N
1.2^N
N^{10}
N^5
N^3
$5N$
Number of nanoseconds since Big Bang
Number of nanoseconds in one day
2
4
8
16
32
64
128
256
512
1024
2048

Different Functions of Time Complexity (V)

	Function \ N	20	40	60	100	300
Polynomial	N^2	1/2500 millisecond	1/625 millisecond	1/278 millisecond	1/100 millisecond	1/11 millisecond
Polynomial	N^5	1/300 second	1/10 second	78/100 second	10 seconds	40.5 minutes
Exponential	2^N	1/1000 second	18.3 minutes	36.5 years	400 billion centuries	a 72-digit number of centuries
Exponential	N^N	3.3 billion years	a 46-digit number of centuries	an 89-digit number of centuries	a 182-digit number of centuries	a 725-digit number of centuries

Reasonable Algorithms

- Examining the previous three slides we find that some algorithms might be reasonable and others not reasonable.
- *Polynomial-time algorithms* are said to be *reasonable.*

Unreasonable Algorithms

- An algorithm that requires exponential or (or super-polynomial) time is said to be *unreasonable*.

Reasonable Vs Unreasonable Algorithms (I)

- Consider N^{1000} and N!.. Which one is better for values lower than 1500?

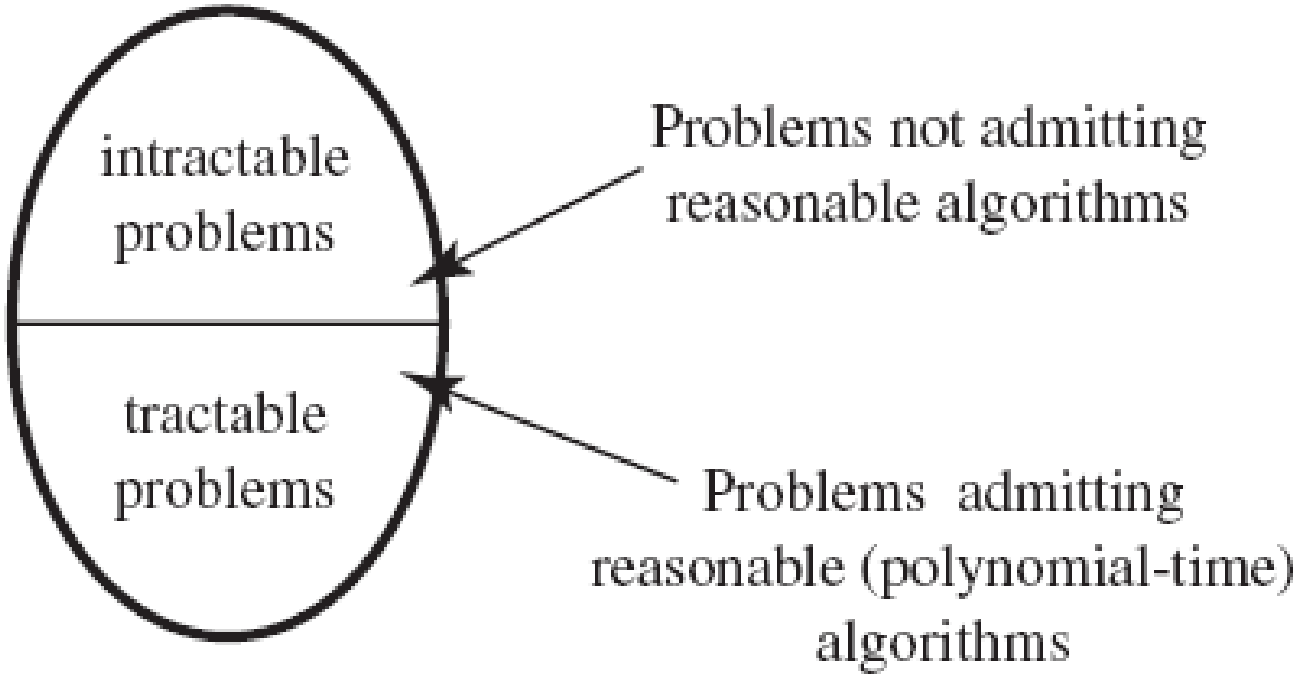
Reasonable Vs Unreasonable Algorithms (I)

- Consider N^{1000} and N!.. Which one is better for values lower than 1500?
- For inputs under 1165, N^{1000} is worse than N!

Reasonable Vs Unreasonable Algorithms (II)

- However, the vast majority of reasonable algorithms that take N^K have a K no greater that 5 or 6.
- Thus, most of the reasonable algorithms are useful and most of the unreasonable algorithms are useless.

Tractable and Intractable Problems



More on the Monkey Puzzle

1. Computers are becoming faster by the week. Over the last 10 years or so computer speed has increased roughly by a factor of 50. Perhaps obtaining a practical solution to the problem is just a question of awaiting an additional improvement in computer speed.

2. Doesn’t the fact that we have not found a better algorithm for this problem indicate our incompetence at devising efficient algorithms? Shouldn’t computer scientists be working at trying to improve the situation rather than spending their time teaching classes about it?
3. Haven’t people tried to look for an exponential-time lower bound on the problem, so that we might have a *proof* that no reasonable algorithm exists?

4. Maybe the whole issue is not worth the effort, as the monkey puzzle problem is just one specific problem. It might be a colorful one, but it certainly doesn't look like a very important one.

Reply point 1

Function	Maximal number of cards solvable in one hour:		
	with today's computer	with computer 100 times faster	with computer 1000 times faster
N	A	$100 \times A$	$1000 \times A$
N^2	B	$10 \times B$	$31.6 \times B$
2^N	C	$C + 6.64$	$C + 9.97$

- Reply to point 2: maybe the students can?
- Reply to point 3: Yes, they've tried, but no one has found that they require *super-polynomial-time*. Best lower bounds: $\Omega(N)$.
- Reply to point 4: It so happens that the "Monkey Puzzle" problem is not alone: There are close to 1000 diverse algorithmic problems like this one. They have the amazing property that if we solve one of them, we solve all of them!!

Problems NP-Complete (I)

- Problems which have unreasonable solutions. But, none are known not to have reasonable solutions.
- No one has proven they require super-polynomial time!!!
- The best-known lower bounds are $\Omega(N)$.
- Upper bounds: exponential

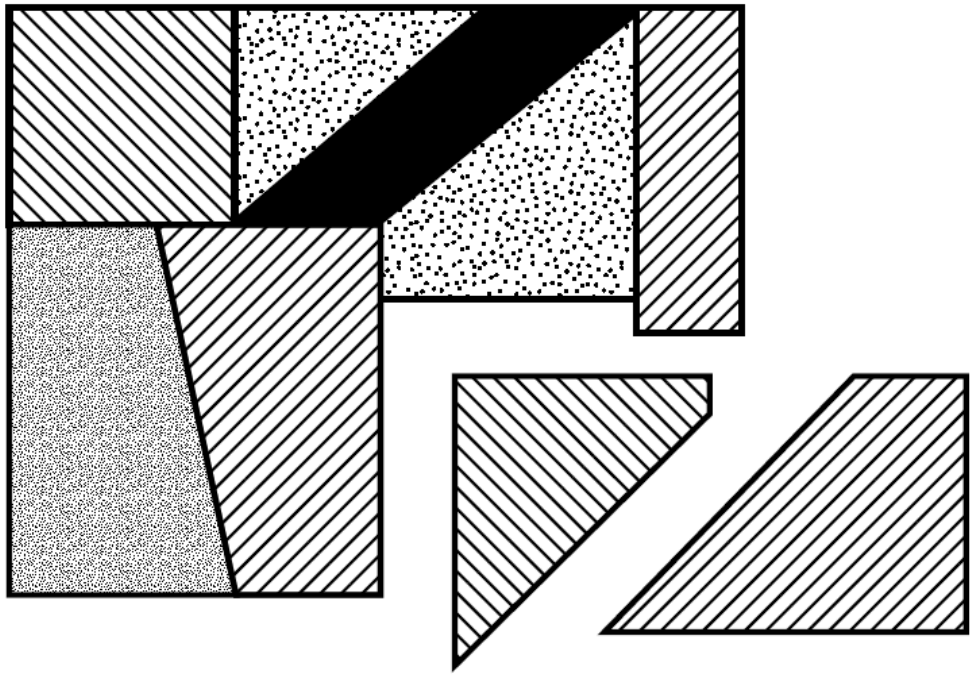
Problems NP-Complete (II)

- They are **complete.** That means that if we solve one of them, we can solve all of them.
- Similarly, if we prove one of them has an exponential lower bound, that applies for all of them!

Some NPC Problems

- Generalized jigsaw or arrangement
- Graph problems
  - TSP (the Decision Version)
  - Hamiltonian path
  - Scheduling and Matching problem
  - Determining Logical Truth
  - Map Colorings
  - Graph Coloring

Airline puzzle

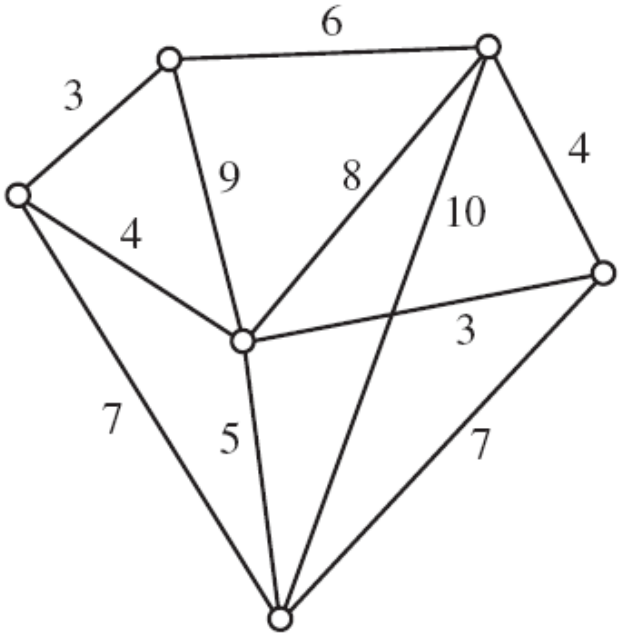


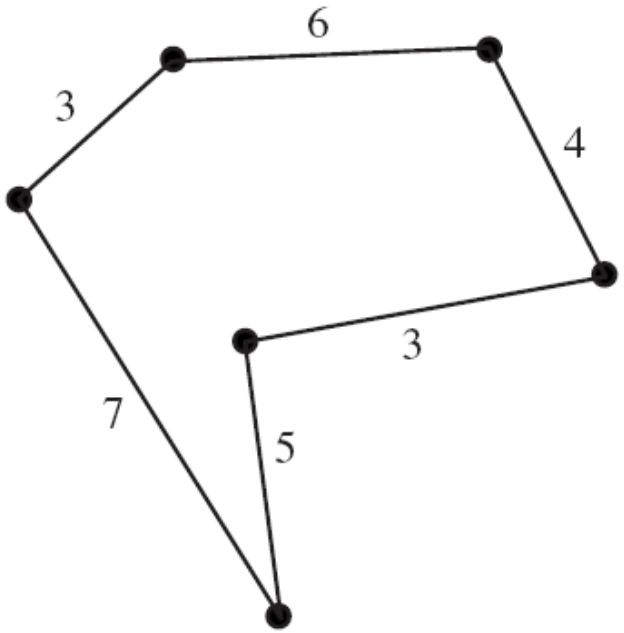
- Can they form a rectangle?

Ordinary Puzzle

- If just one piece can fit in a given place, the time complexity is quadratic.
- If several pieces can fit in a given place is NPC because partial matches may not lead to a complete solution.

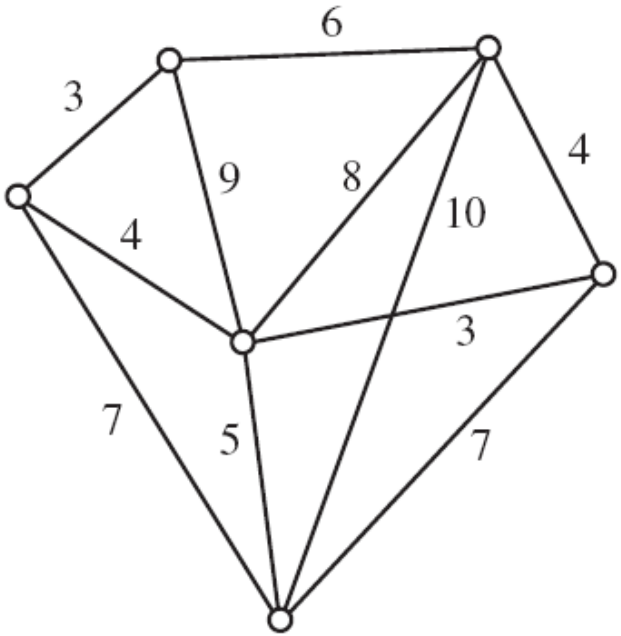
Decision Version of the Travelling Salesman Problem

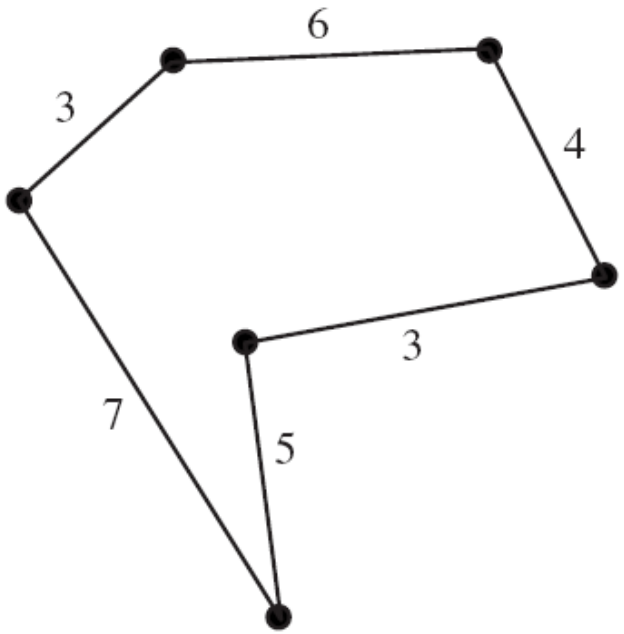




Given a list of cities and the distances between each pair of cities, and a length L, determine whether the graph has any tour shorter than L.

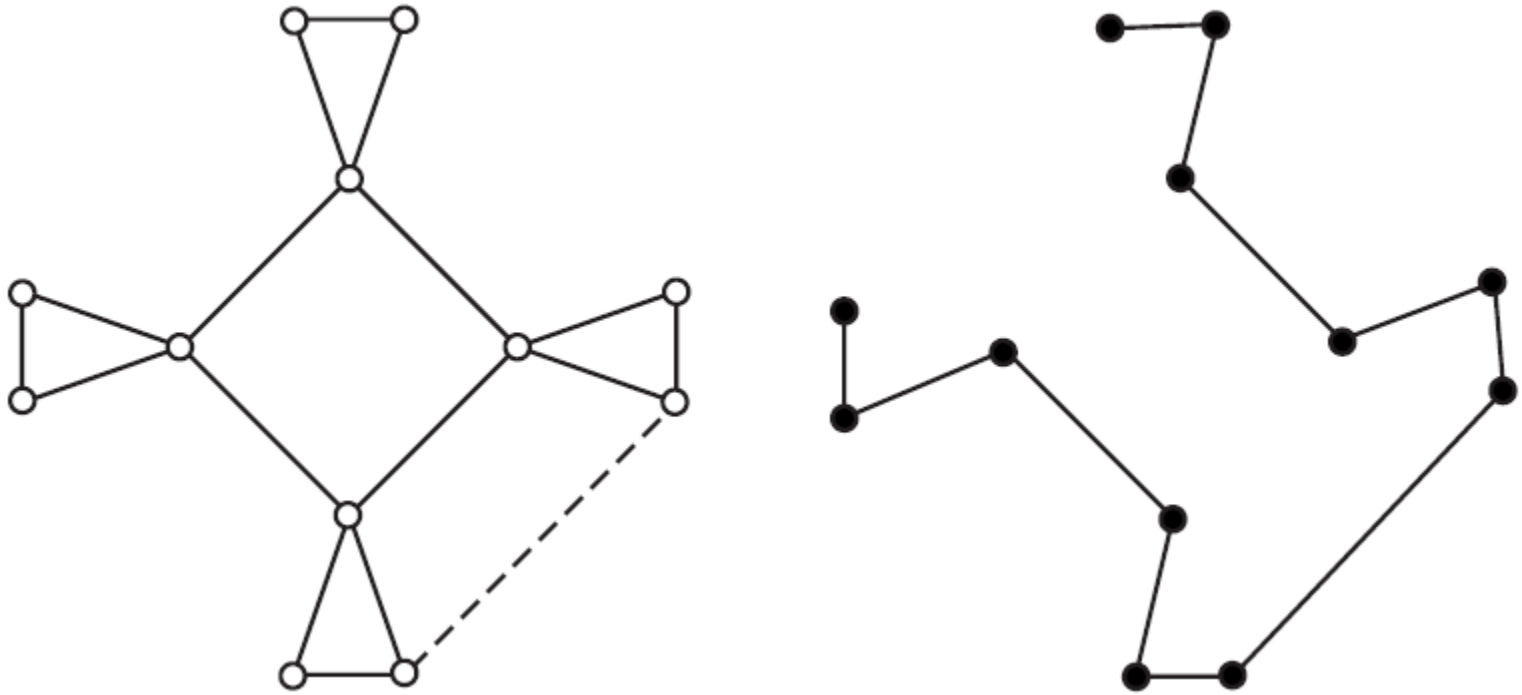
Traveling-Salesman Problem





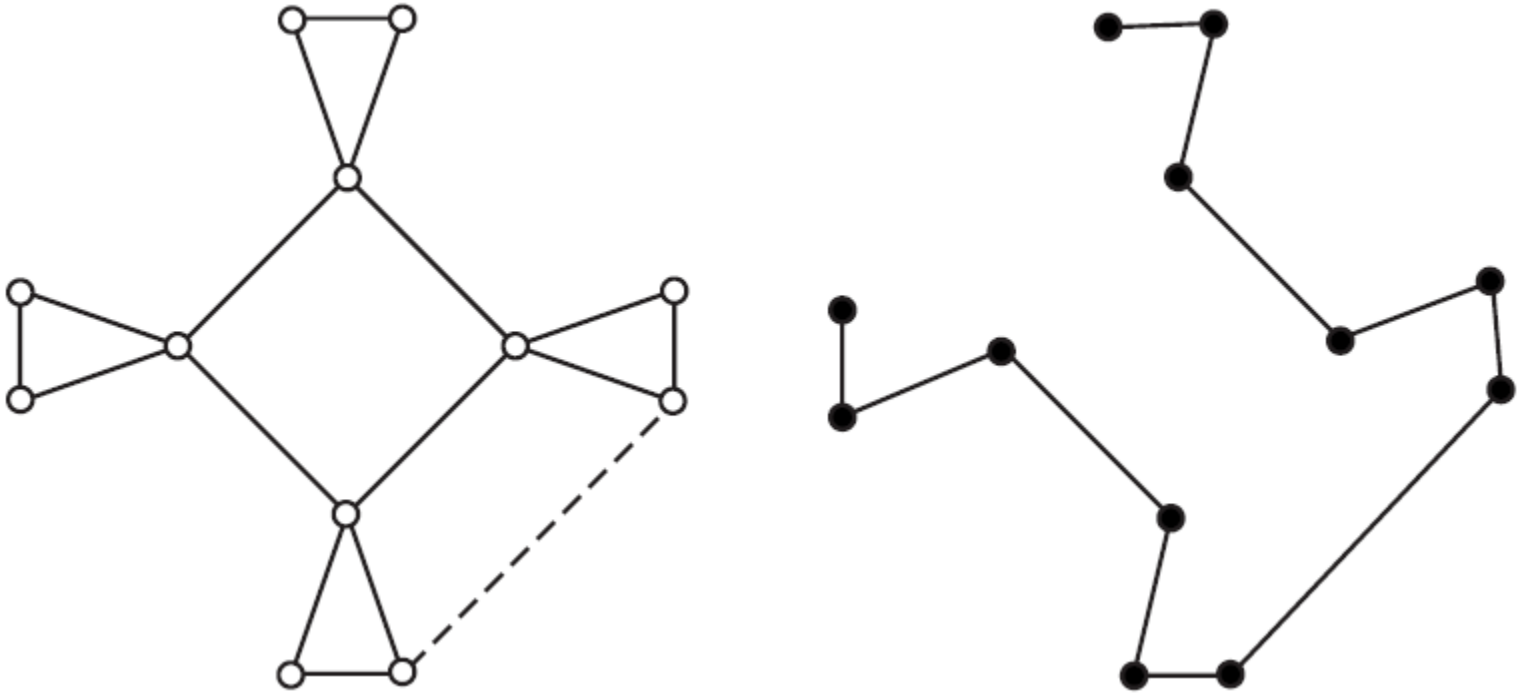
What is the naive solution? What is its complexity?

Hamiltonian Path



Path in an undirected or directed graph that visits each vertex exactly once.

Hamiltonian Path



What is its complexity?

Other problems

- Satisfiability: Given a logical sentence, we want to know whether it is possible to assign “true” and “false” to basic assertations so that the entire sentence is true.
- Map Colorings
- Graph Coloring

Solution to NPC problems

- Explore all options!
- Try to extend *partial matches* to find whether they lead to *complete matches*.

Certificate (I)

- In a NPC decision problem, it's hard to determine if the answer is yes or no.
- However, with an evidence that proves that the answer is yes, it is easy to verify that it is indeed yes.
- Such evidence is called **certificate.**
- They are short (polynomial in N).

Certificate (II)

- Give examples of certificates for
  - The Monkey Puzzle problem
  - The TSP problem
  - The satisfiability problem.
  - The map coloring problem
- What is the complexity to verify that the answer is yes, using such certificates?

Nondeterministic Algorithms

- They use such magic coins or oracles.
- They magicly guess which of the available options is better rather than having to employ some deterministic time to go through all of them.
- Think of a nondeterministic algorithm as an abstract notion, not as a realistic goal.

Nondeterminism & NPC

- Every NPC problem has a polynomial time non-deterministic algorithm.
- This fact can be proven by showing that the short certificates correspond to the polynomial time solutions.
- NPC:
  - NP: Nondeterministic polynomial time
  - C: Complete

Complete

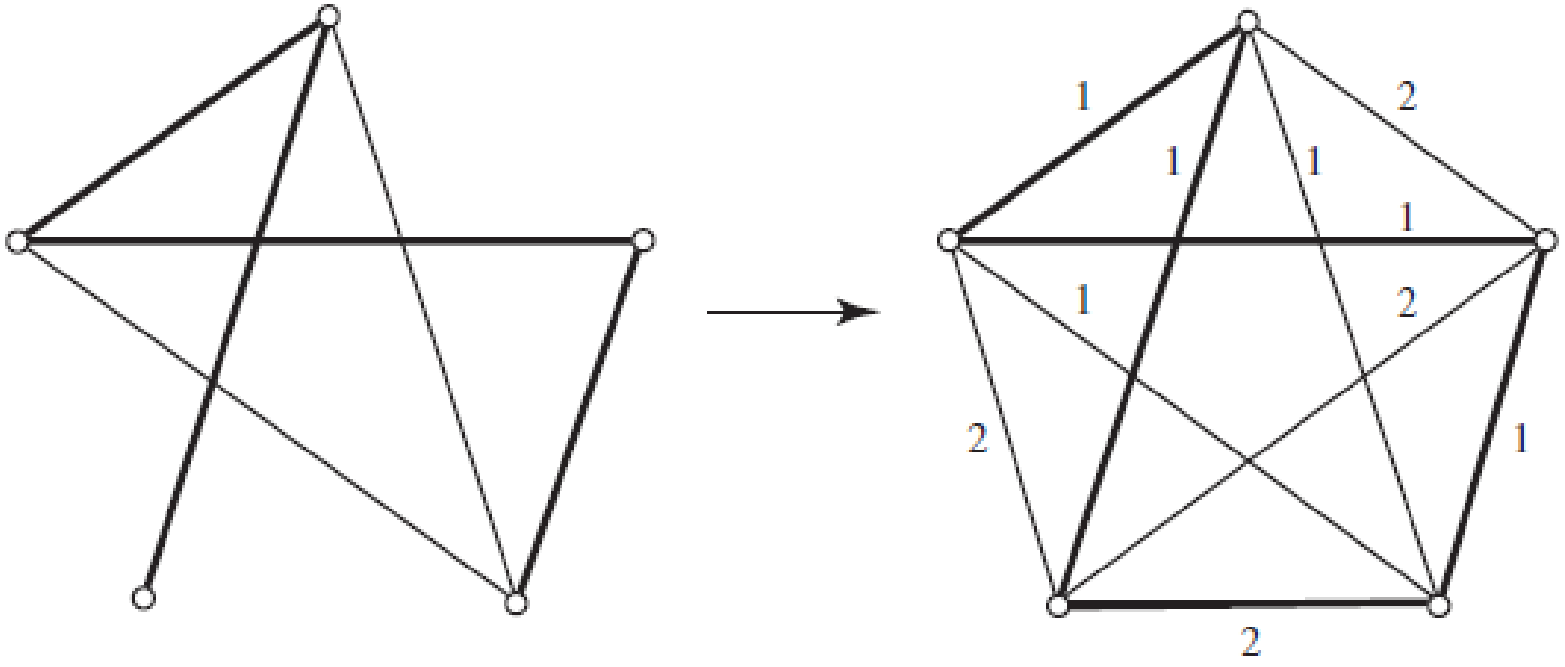
- Either all of them are tractable or none of them is!
- If someone proved an exponential-time lower bound for any NPC problem, then no NPC problem could be solve in polynomial time,
- If someone found a polynomial time algorithm for an NPC problem, there would be polynomial-time algorithms for all of them.

Polynomial-Time Reduction (I)

- Given two NP-Complete problems, a polynomial-time reduction is an algorithm that runs in polynomial time and reduces one problem to the other.
- If problem **X** can be polynomically reduced to problem **Y**. It means that **X** is no worse than **Y**.

Polynomial-Time Reduction (II)

- Example



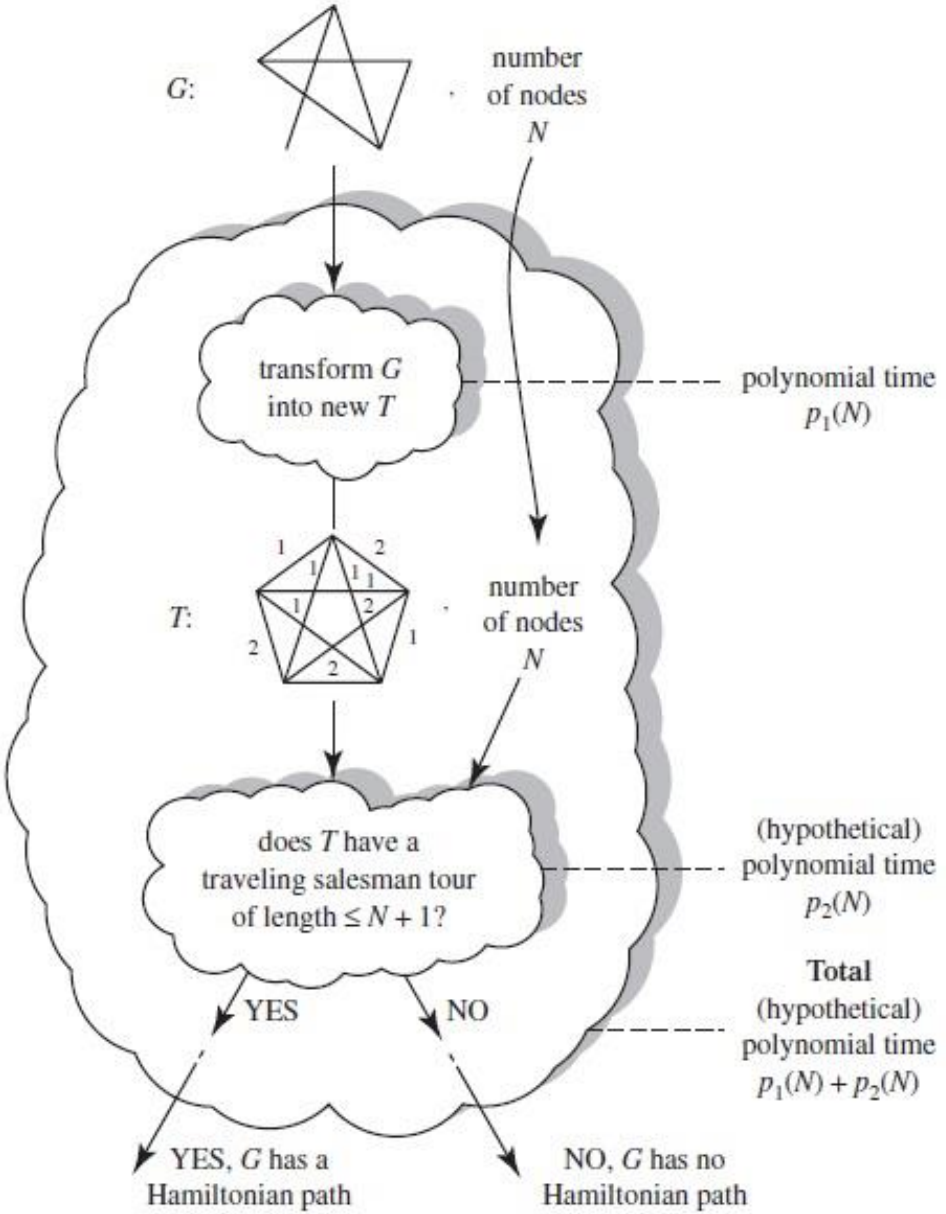
G

Hamiltonian path emphasized

T

Traveling salesman tour of length 6 emphasized

Polynomial-Time Reduction (III)



Exercise

- How would you polynomically reduce the 3-coloring problem to the satisfiability problem?
  - N countries C1, C2, …, CN
  - Colors R (red), B (blue), Y (yellow)
  - Propositional sentence F

Solution

- 3N assertions… why?

$$((C_I\text{-}is\text{-}R \;\&\; \sim C_I\text{-}is\text{-}B \;\&\; \sim C_I\text{-}is\text{-}Y)$$

$$\vee\,(C_I\text{-}is\text{-}B \;\&\; \sim C_I\text{-}is\text{-}R \;\&\; \sim C_I\text{-}is\text{-}Y)$$

$$\vee\,(C_I\text{-}is\text{-}Y \;\&\; \sim C_I\text{-}is\text{-}B \;\&\; \sim C_I\text{-}is\text{-}R))$$

$$\sim((C_I\text{-}is\text{-}R \;\&\; C_J\text{-}is\text{-}R)$$

$$\vee\,(C_I\text{-}is\text{-}B \;\&\; C_J\text{-}is\text{-}B)$$

$$\vee\,(C_I\text{-}is\text{-}Y \;\&\; C_J\text{-}is\text{-}Y))$$

- What is the time of the reduction?

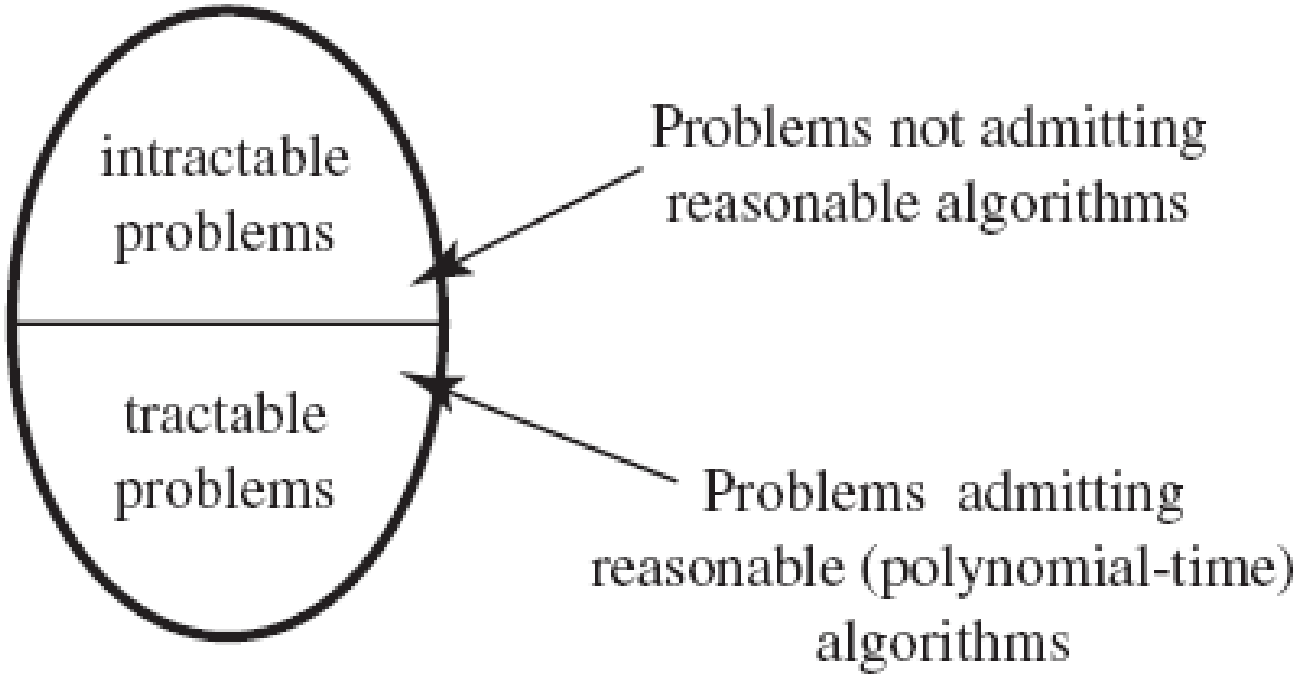
Showing that a problem is NP-Complete (I)

- Cook's remarkable theorem proved that there exists **NP-complete** problems. In particular, he demonstrated the SAT was **NP-complete** (1971).
- Cook's theorem makes showing a problem is **NP-complete** easier.

Showing that a problem is NP-Complete (II)

- New problem R
- You must prove that you can
  - Reduce R to a NP-Complete problem Q (or show a certificate/non-deterministic magical algorithm).
  - Reduce a NP-Complete problem S to R (or use a theorem)

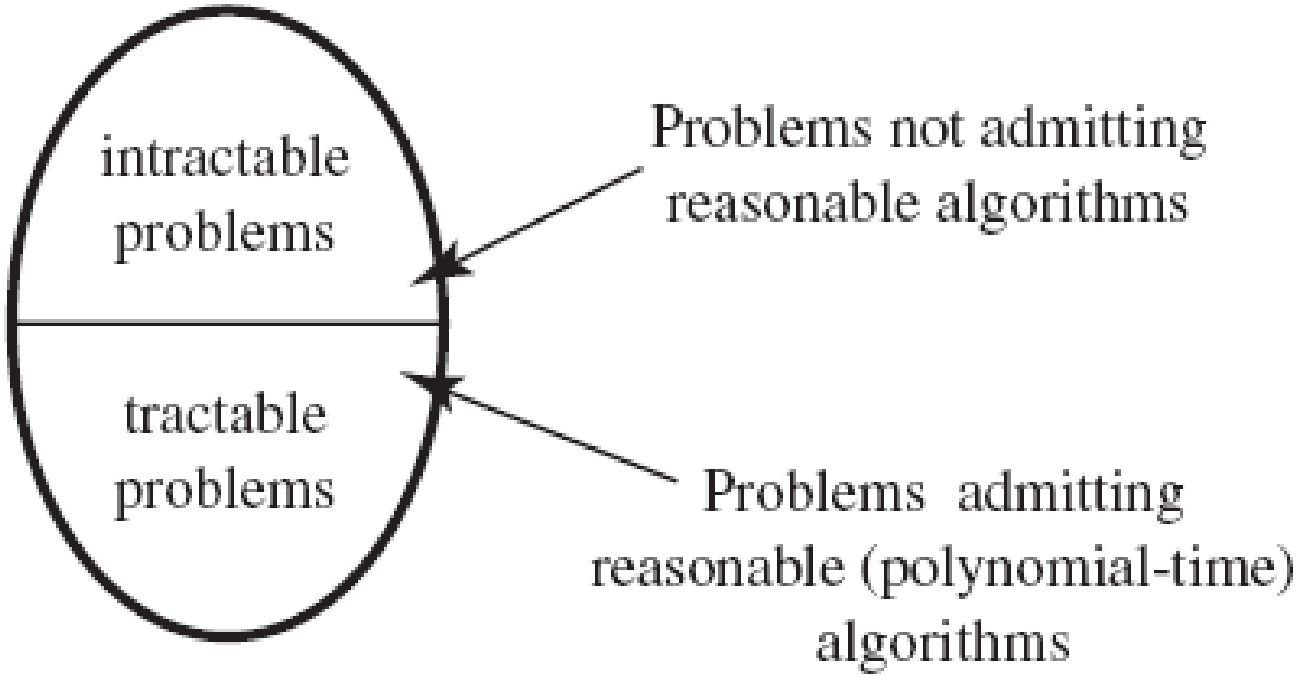
Where are the NP-Complete problems in the sphere?



Class P

- The class **P** is comprised by the set of problems that are tractable, i.e. problems that can be solved by a deterministic-polynomial time algorithm.

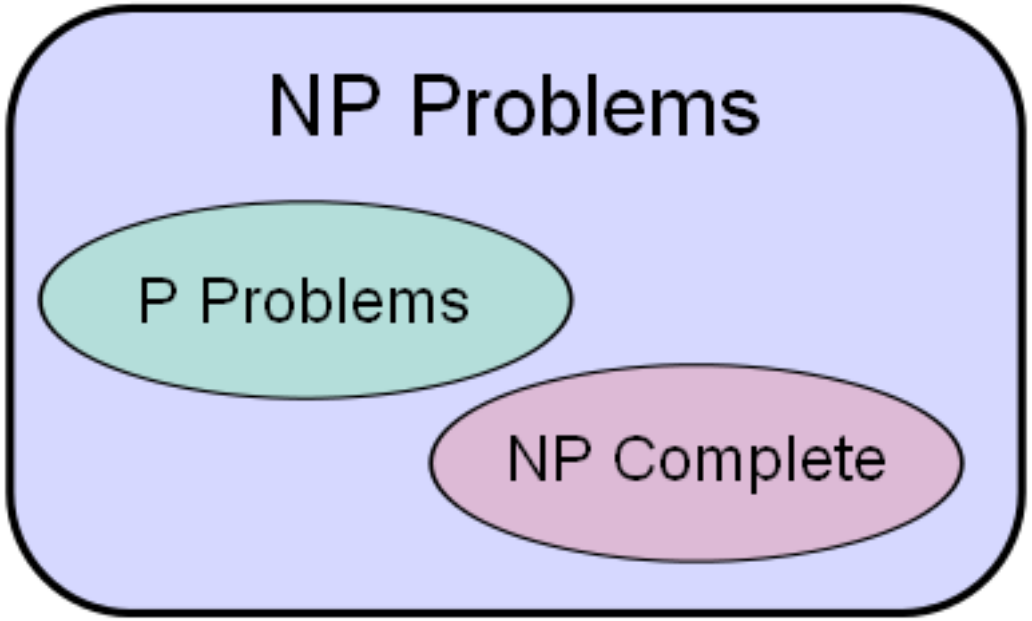
Where are the P problems in the sphere?



Class NP (I)

- NP is the set of all decision problems for which the instances where the answer is "yes" have efficiently verifiable proofs of the fact that the answer is indeed "yes."
- More precisely, these proofs have to be verifiable in polynomial time.
- The hardest problems in NP are NP-Complete problems.

Class NP (II)



- P = NP ? First asked in 1971
- Is there NP-Intermediate?

P = NP? (I)

- Its clear that is $\mathbf{P} \subset \mathbf{NP}$, but $\mathbf{P} = \mathbf{NP}$?
- No one has yet demonstrated a problem that is in **NP** but not in **P**!
- To prove **P** = **NP** requires us to show that every problem in **NP** can be solved by a deterministic polynomial time algorithms. Nobody has done this either!

NP-Intermediate Problems (I)

- Set of problems that are in the complexity class **NP** but are neither in the class **P** nor **NP-complete**.
- The class of such problems is called **NPI**.

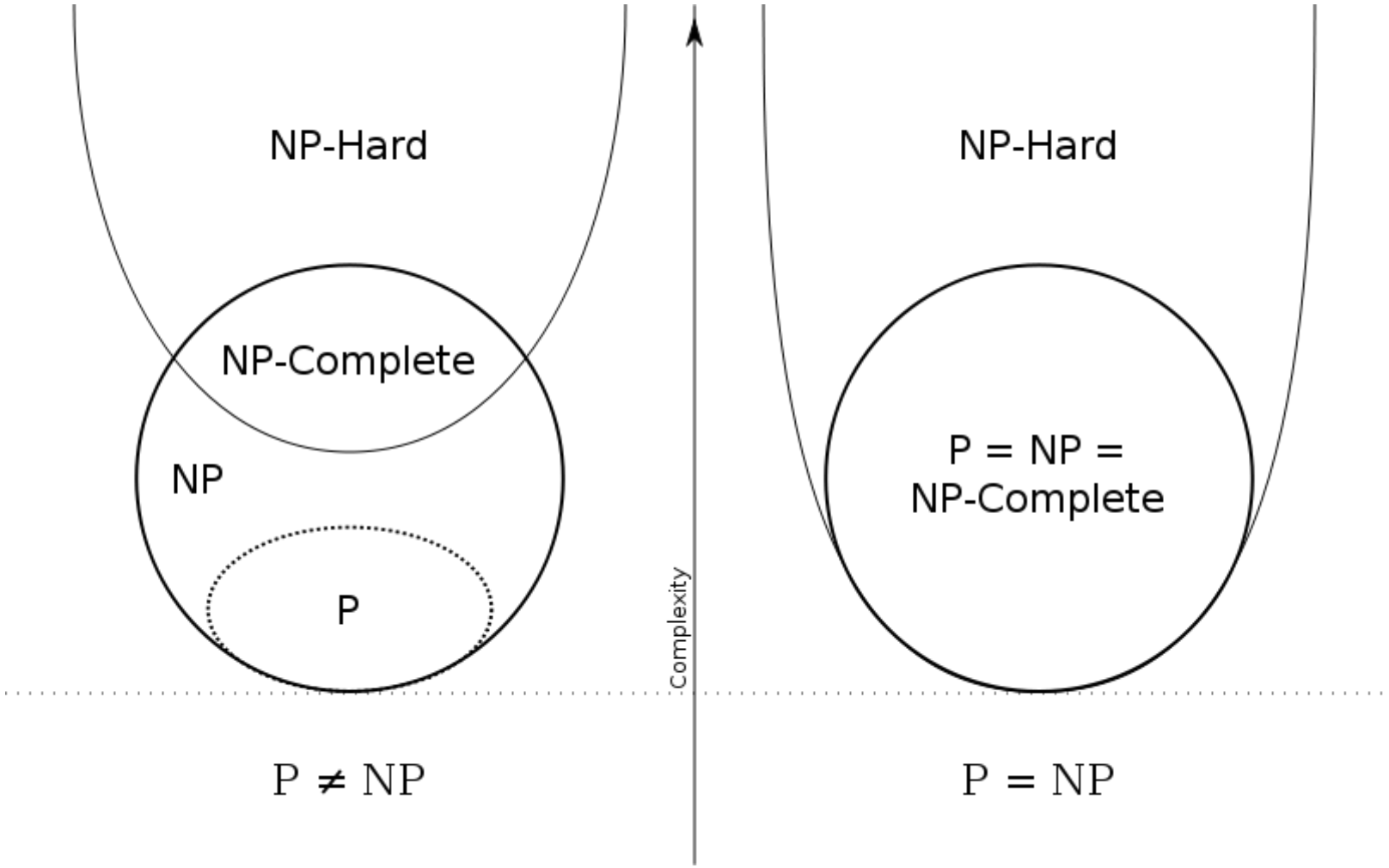
NP-Intermediate Problems (II)

- **Ladner's theorem**, shown in 1975, is a result asserting that:
  - If **P ≠ NP**, then NPI is not empty; that is, NP contains problems that are neither in P nor NP-complete.
  - **P = NP** if and only if NPI is empty.
- NPI Candidates:
  - graph isomorphism problem
  - computing the discrete logarithm.

NP-Hard Problems (I)

- At least as hard as the hardest problems in NP.
- A problem X is NP-hard if and only if there is an NP-complete problem L that is polynomically reducible to X.
- This means that no problem in **NP** is harder than X.

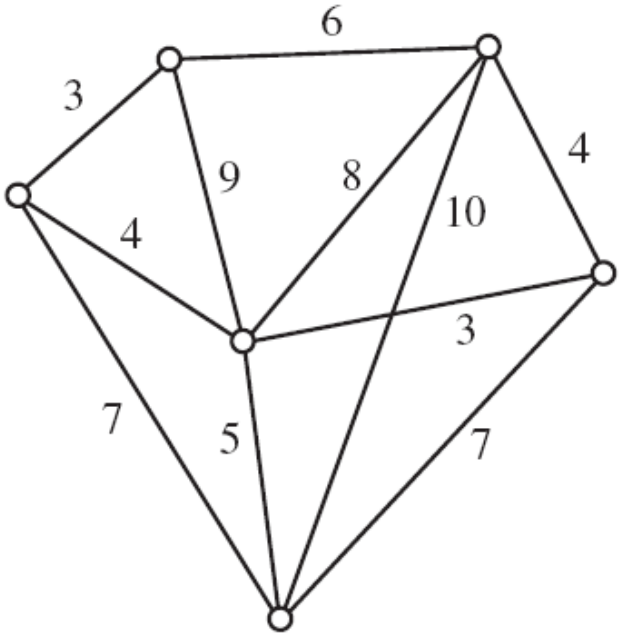
NP-Hard Problems (II)

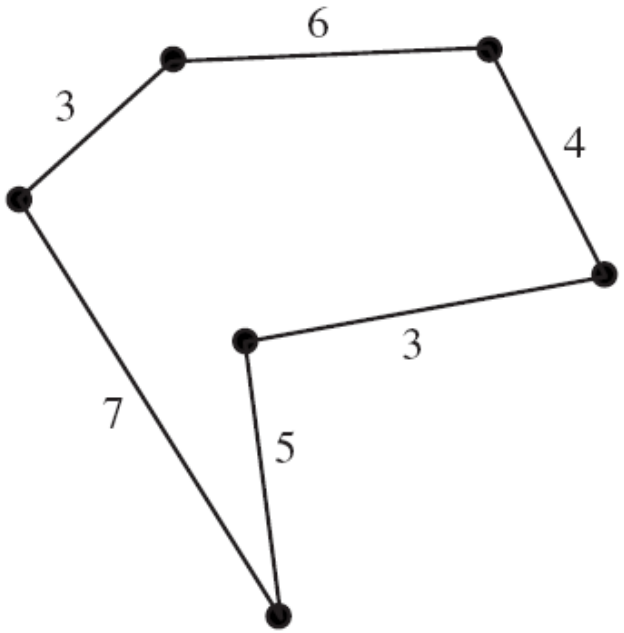


NP-Hard Problems (III)

- A problem X is called an **NP-complete** problem if (1) it belongs to **NP** and (2) it is **NP-hard**.
- Note that, if a **NP-complete** problem can be solved by a deterministic polynomial algorithm, then **P = NP**.
- Also, if any **NP-hard** problem is shown to be in **P**, then **P = NP**.

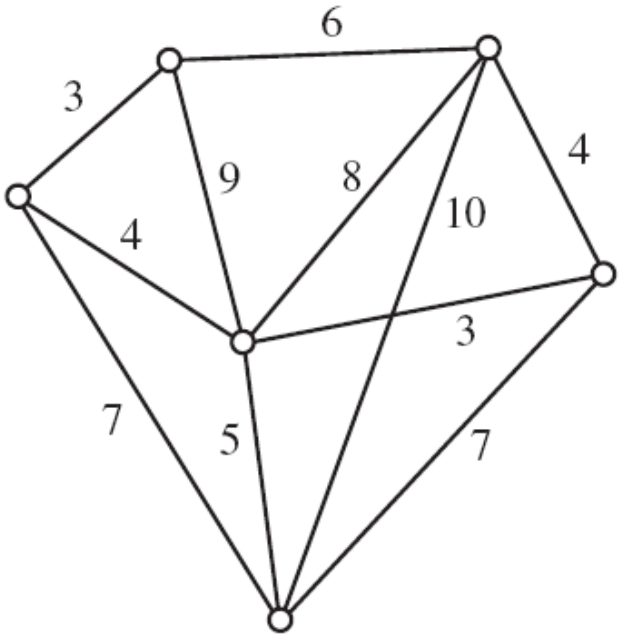
Example of an NP-Hard Problem: TSP

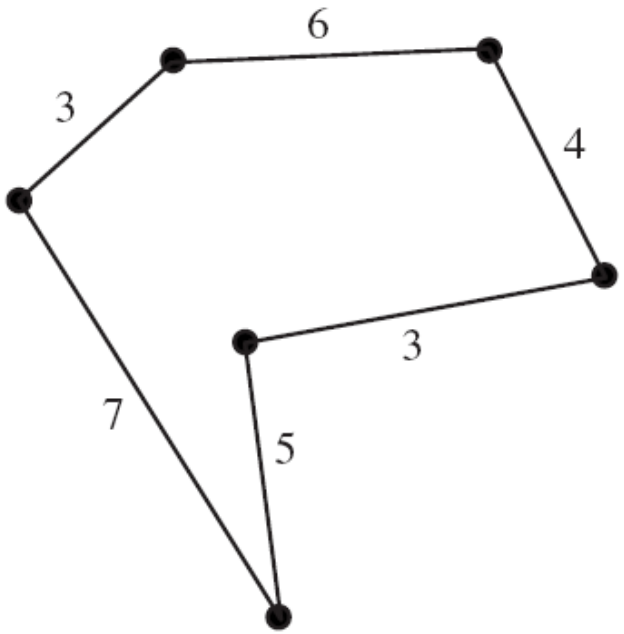




Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city!

Traveling-Salesman Problem





What is the naive solution? What is its complexity?

Optimizations Problems that are NP-Complete

- **NP-complete** problems are usually taken as the *decision* problem (yes/no) version of *combinatorial optimization* problems. E.g. TSP.
- Finding an optimal tour cannot be tractable without the yes/no version being tractable too.
- If the decision problem is NP-Complete, the original problem is likely to be NP-Complete or worse.

Imperfect Solutions to Hard Problems (I)

- But, humans manage to solve these problems all the time in short time!
- **Approximation algorithms** find solutions less than perfect, yet of practical value.
- Some of them are guaranteed to be close the optimal solution.

Imperfect Solutions to Hard Problems (II)

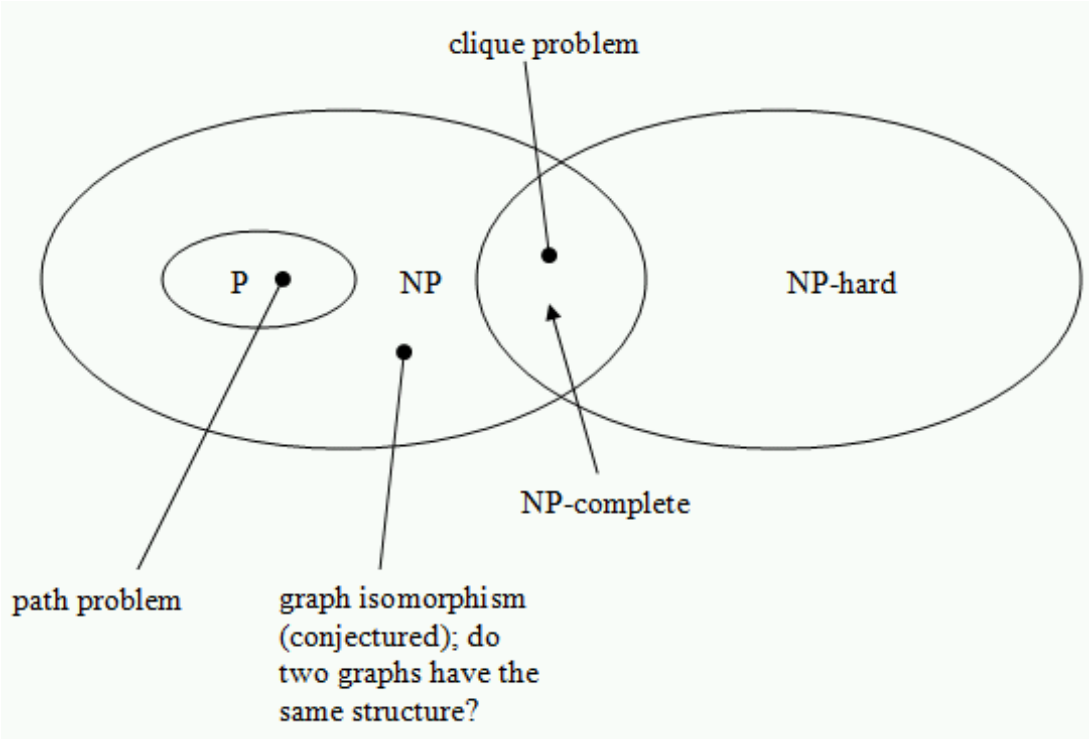
- Some of them are guaranteed to be close the optimal solution.
  - Ex: Cubic algorithm that always produces a result at most 1.5 worse than the optimal (for TSP).
- Some of them produce a solution very close to the optimal solution almost always.
  - Ex: Algorithm that parts the graph and combines local solutions.

Provably Intractable Problems

- We are not able to *prove* that problems in **NP** have no tractable solution
- There are problems, however, such as the Towers Of Hanoi, that can be shown to have an exponential lower bound and that have no *certificate*.

Complexity Classes

- Theoretical Computer Scientists talk about a hierarchy of complexity classes.



- These classes are for both **time** and **space** requirements.

Research on Complexity Classes

- Some theorists work with questions such as: *Is P = NP?* or *Can we refine the complexity classes?*
- Others work to develop approximate solutions or heuristic solutions or alternate models of computation: genetic algorithms, neural networks, etc..

Bibliography

- Harel D., Feldman Y. *Algorithmics: The Spirit of Computing (3rd Edition)*. Addison Wesley; 2004.
- Garey, M. R. & Johnson, D. S. (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman.