# Problems and Solutions

# Decidability and Complexity

# *Algorithm*

## *Muhammad bin-Musa **Al** **K**h**w**a**rism**i (780-850)*

- 825 AD System of numbers "***Algorithm or Algorizm***"
  - On Calculations with Hindu (*sci* Arabic) Numbers

    (Latin Algoritmi de numero Indorum)
- 830 AD Calculation by Transposition and Cancellation ***"Algebra"***
  - Hidab al-jebr wa'l-muqabala

# Kinds of problems for which we seek solutions

- Decision
  - is a >b ?  yes or no?
  - *is h* true or is *h* false
  - The binary digit is 1 or 0
- Function (Optimization problems can be a subset)
  - Area = $\pi r^2$
  - Shortest route that touches each node in a network exactly once.
- Enumeration
  - prime numbers between 10,000 and 10,500 ?

# Kinds of problems

- By and large, we will focus only on decision problems.
  - Function problems and optimization problems can be re-formulated as decision problems
  - We base our entire theory of complexity on decision problems

Note*: decidability* is not the same thing as a decision problem

# Solutions

WHAT IS A SOLUTION TO A DECISION PROBLEM ?

- Concrete notion:
  - Algorithm maps to {0,1}
- Abstract notion:
  - Enter an accepting state and HALT

# Solutions

- Can happen
  - Problem is decidable
    - Sometimes fast
    - Sometimes after a very, very, very long time

- Can never happen *a priori*
  - Problem is ***undecidable***

# Suppose the computer is working away on the problem ….

Is there an oversight program that can tell us whether

- our specific program is working on a solvable problem that takes a very long time to decide?

or……

- is the problem one that can never be decided, and the computer will never halt?

Answer may not be what you might expect……

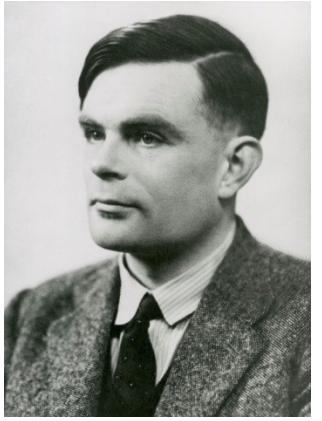
There may be a program that can tell us that our specific program of interest will halt or not

**BUT**

**There is no program that can tell us whether ANY program will halt or not**

# A proven undecidable problem

- The strategy is to design an oversight program that, given an input program, arrives at a contradiction.
- Because the answer is a contradiction, such a program cannot exist
  - The problem is therefore  undecidable
  - This is called the Halting Problem, and the contradiction proved by Alan Turing



# A proven undecidable problem

The idea of the proof is to feed output, reversed, back into the input

- Example

  The barber shaves everyone who does not shave himself. Who shaves the barber?

- Another example

  This sentence is false

- Another example

  Message #1: Message #2 to follow

  Message #2: Ignore Message #1

# The Halting Problem

## …a bit more detail

Does any program input to itself halt?

This requires a decision yes/no, if one can be made

It turns out that no decision can be made; the problem is ***undecidable***

Let S be a program and let X be an input to S, where S is the most sophisticated design that is known (a Turing Machine)

If X halts on itself as input, then S loops,
Else If X does not halt on itself, then S halts

} The reversal razzle-dazzle

Now…

Input S, instead of X, into S

Either

S halts, so the output of S is no halt, despite the assumption that it did halt

or

S does not halt, forcing S to halt, under the assumption that it did not

# Decidable Problems

# Complexity of a decidable problem/solution

Complexity is a measure of the number of elementary operations, say, additions, to decide a decidable problem.

- Complexity is usually stated not in ops, but in time
  - The notation 'big O' ($\mathcal{O}$) describes the order of time involved, *i.e.,* its asymptotic behavior
- Complexity can be alternatively be expressed computational space
- Typically there is an upper and lower bound on complexity

# Complexity

Let **n** be the size of the problem and **k** be a constant

- Polynomial (P) complexity $n^k$
  - Super-polynomial: Worse than polynomial but not really exponential $n^{\log n}$
- Exponential complexity $k^n$
  - Super-exponential: worse than exponential $n^n$

Note, however, that there is a solution in finite time

# Complexity

- Problems that require Polynomial time/space to solve are said to be *tractable*
  - However there may be instances which, although the complexity is polynomial, require practically infinite time to solve $a^{10^{10^{10^{10}}}}$
- Problems that require exponential time/space to solve are said to be *intractable*
  - However, they are decidable and can be decided in finite (albeit very long) time

# Computational Complexity- Tractable problems

1. Find the sum of 2 numbers
2. Find the sum of 20 numbers
3. Find the sum of 37 trillion numbers

Same algorithm for all 3 instances, with linear complexity.

Would use a DO-loop

Can build a more efficient algorithm for #1 (and perhaps #2), specific instances of the problem, but the solution for these specific instances is not reasonably generalizable

# Complexity -Tractable Problems

Sometimes efficiency can reduce complexity

- Linear search $\mathcal{O}(n)$
- Binary search $\mathcal{O}(\log_2 n)$

Efficiency can sometimes depend on the instance of the problem.

Extreme example: If the solution for the search of an ordered list of 16 integers happened to be 1, linear search would be one cycle, where binary search would take $\log_2 n$ or 4 cycles. On the other hand, suppose the answer is 11. The linear search would take 11 cycles, the binary search would take 4.

Sometimes if we know the character/scope of the solution set, we can better choose our algorithm

# Complexity -Tractable Problems

Efficiency Improvement: another GREAT example is the Fourier Transform $F(\omega) = \frac{1}{2\pi} \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt$

- Brute force solution (all frequencies, all data points) is $\mathcal{O}(n^2)$
- Special instance (FastFourierTransform) of the problem where the number of data points is a power of 2: $\mathcal{O}(\log_2 n)$
  - NB that the solution is exact at the corresponding harmonics

Note that both solutions in this case are polynomial, but when the FFT was developed in the era of slow computers, with large n, the gain was substantial. In this case we chose the algorithm based on the specific properties of the problem

# Proven intractable problems

- A proven intractable problem: Does a regular expression with exponentiation denote all strings over its alphabet?
  - It has been PROVEN that the solution is exponential, and PROVEN that there is no polynomial solution.
- Towers of Hanoi
  - Proven that solution is exponential $\mathcal{O}(2^n\text{-}1)$

Very, very, very  few problems fall into this class of <u>proven</u> exponential complexity

# Problems thought, without proof, to be intractable

- Many, many seemingly problems seem obviously exponential, and, as yet, they lack a polynomial solution and are solved in exponential time (still finite, by the way)
  - In many problems, including a great many that we will study relating to Computational Molecular Biology, the problems appear to be exponential and as yet there has been no polynomial solution found. For example, it is not uncommon in optimizations to be looking at permutations that are super-exponential

$$\Theta\left(\frac{n}{e}\right)^n$$

# The Lingo of Complexity

Consider this problem:

$f = (A \vee B) \wedge (-C \vee D)$

where *f*,A,B,C,D are Boolean variables.

[The size of the problem is 4]

Is there an instance where f is true?

Trivia note: Notice that the problem is expressed as the conjunction of disjunctive clauses (2, in this case). The is called the Conjunctive Normal Form. The structure of this problem is not relevant to the solution of this particular problem but is important in generalizing the problem.

# Complexity

- Given any specific instance (*i.e.*, given specific T/F values for A,B,C,D), we can easily (*sci* in polynomial time) decide if ***f*** is true or false

- But if we are not given a specific instance, we would need to look at all 16 possibilities of A,B,C,D configurations, evaluating each instance in polynomial time.

# Complexity

- Or, perhaps we could make a lucky guess!
  - If we guessed correctly, then the guess would be polynomial, the evaluation would be polynomial, and therefore the entire decision would be polynomial
- In computer-speak, the correct word for ‘guess’ is ‘non determinism’

# Complexity

But now suppose that the problem were

$f = (A \vee B) \wedge (-C \vee D) \wedge (-E)$

[The size of the problem is now 5]

- Solution requires either enumeration of all values of independent variables ($2^5$)

or

- a lucky guess

Verifying the solution might take a little longer, but not much

# Complexity

Extending the thought, suppose that the problem were

$f=(A \vee B) \wedge (-C \vee D) \wedge (-E) \wedge (G)$

[The size of the problem is now 6]

In order to *satisfy* the requisite variables to obtain f=true,

- The solution requires either evaluation of all 64 permutations of independent variables ($2^6$)

  or

- a lucky guess

The difficulty of this **SAT**isfiability problem (SAT) appears to be exponential, since the size of the problem is in the exponent of $2^n$

However, *verifying* the solution relates to how many variables, and grows as a polynomial (in this case, linear) function

# Complexity

Suppose we did guess…

Let's guess A=t,B=f,C=t,D=f,E=t,G=t

Wow! A lucky guess, because instead of enumerating 64 choices, we got it right on one guess (although there are certainly other guesses that would have ***satisfied***)

Q. How do we know it is a good guess (verify)?

A. Because we had to plug in the values for A,B,C,D,E,G and evaluate ***f***

Q How long did that take?

A. Polynomial time to verify

So, our solution was

- Non-deterministic ("N") solution (*i.e.* a guess or prior knowledge)*
- **Verifiable** (not the same thing as solvable) in **polynomial** ("P") time

Thus we have a problem class "Nondeterministic Polynomial", or NP", which has a nondeterministic proposed solution and which can be verified as a correct solution (or not) in polynomial time

BUT we don't know that the general solution can be found in polynomial time; right now it sure looks like it is exponential

In fact, we have no known algorithm to solve (complete solution) the SAT problem in polynomial time, although it is remotely possible, but highly unlikely, that one may exist.

*Can also mean modeled on a nondeterministic Turing machine, or, equivalently, computed by a nondeterministic algorithm

# NP problems

- **N**onDeterministic* (after original Turing Machine concept)
- *Verified* in **P**olynomial time
  - Does not necessarily mean *solved*, but rather a proposed solution checked (certificate)

Note that every NP problem is *decidable*. This is a key concept.

*It has been suggested that the term VP (verifiable in polynomial time) might be more apt

# A Fundamental Question

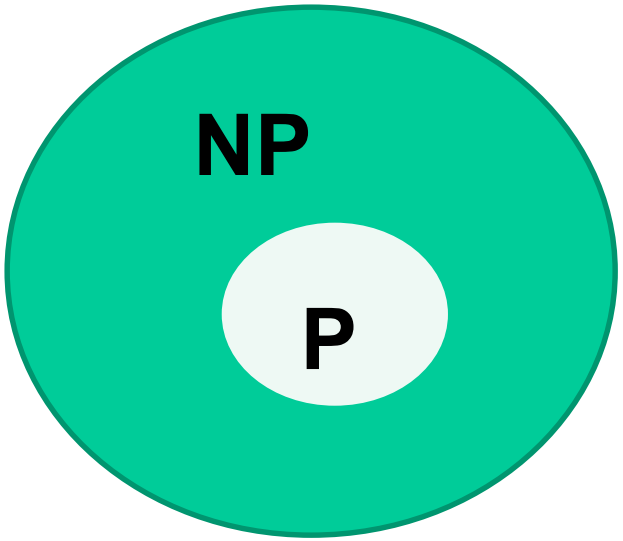
Remember P problems also fit the definition of NP, so….

- Is P is a proper subset ($P \subset NP$)

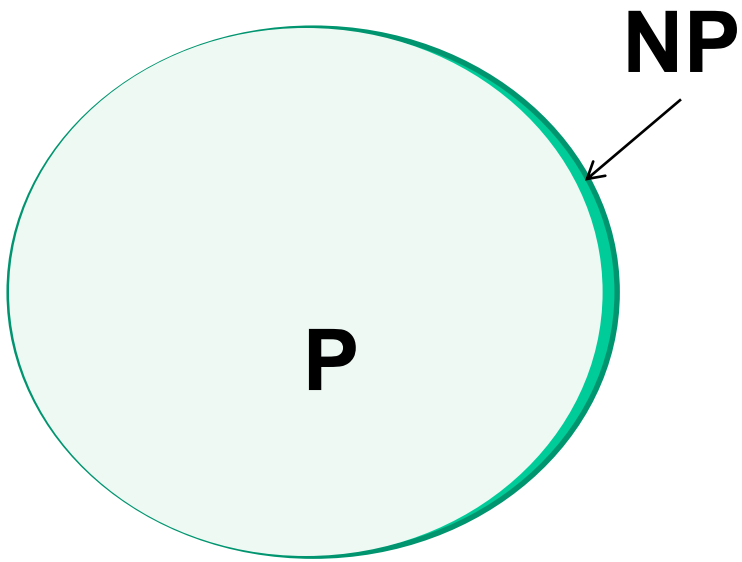
  or

- Can ***all*** problems in NP be ***solved*** in P time ($P=NP$)
  - If that is the case, then other NP problems such as TSP, permutations, factorization, *etc* all must have efficient solutions. These solutions have never been found and do not likely exist

# P and NP




$P \subset NP$




P=NP

# One more new idea…
# Polynomial Reduction

Think of a function that transforms one problem into another

If that transformation can turn the problem into, say, SAT,

and

if that transformation can do it in polynomial time

then

the transformation is called a ***polynomial reduction***

# Down the Rabbit Hole of Infelicitous Terminology…

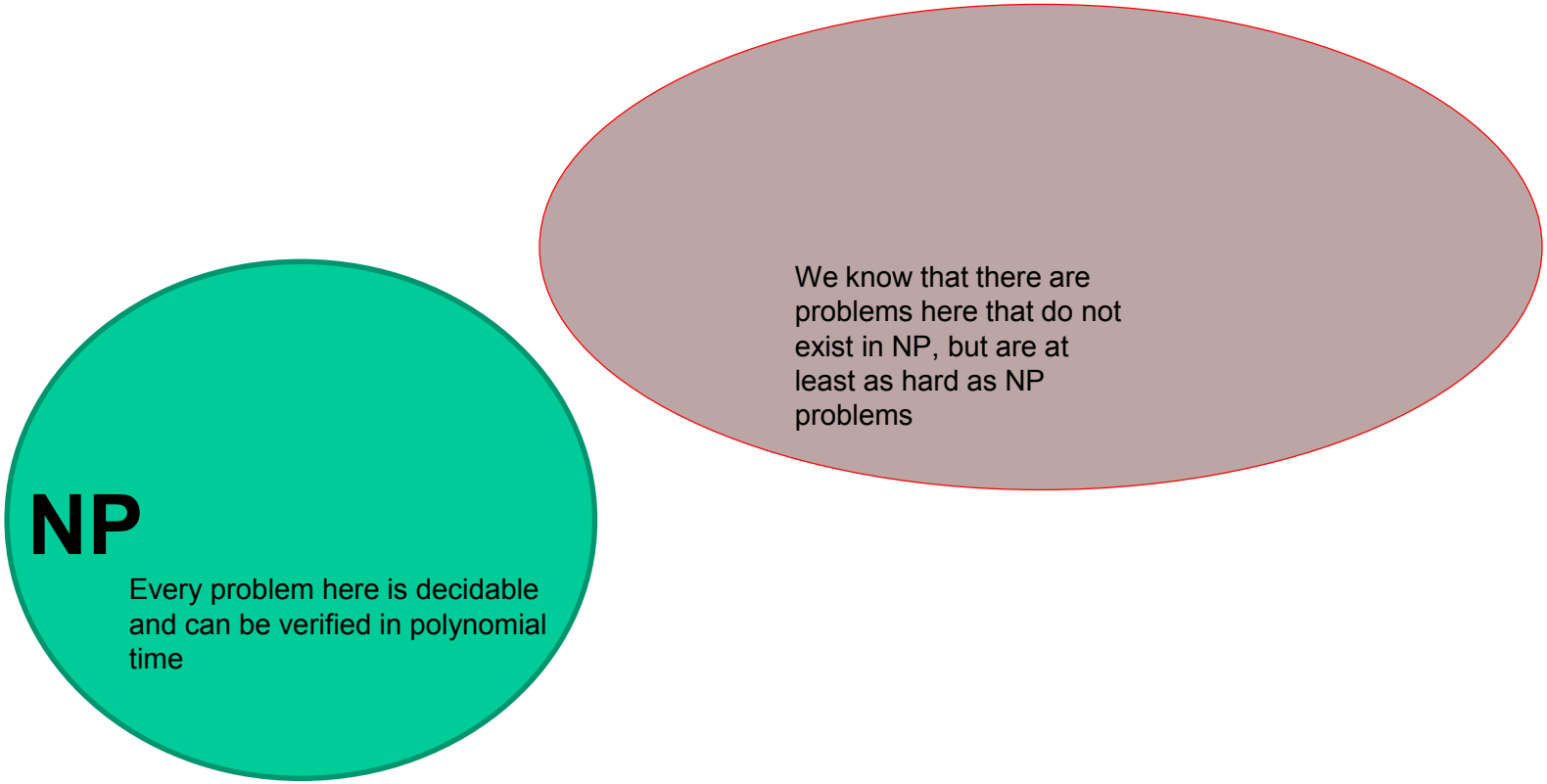
Imagine a class of problems NPH that are at least as hard as NP problems. Some are decidable, some not

- If every problem in NP can be reduced to a problem $x_i$ such as, say, SAT, then {x} are in NPH
- Other problems, not necessarily in NP, are at least as hard as NP problems and would also belong in NPH, *e.g.* The Halting Problem and other non decidable problems*

The problems in the set NPH are called **NP-Hard**

**e.g.* SAT type problems using both universal and existential quantifiers

# NP and NP-Hard



# Further down the Rabbit Hole

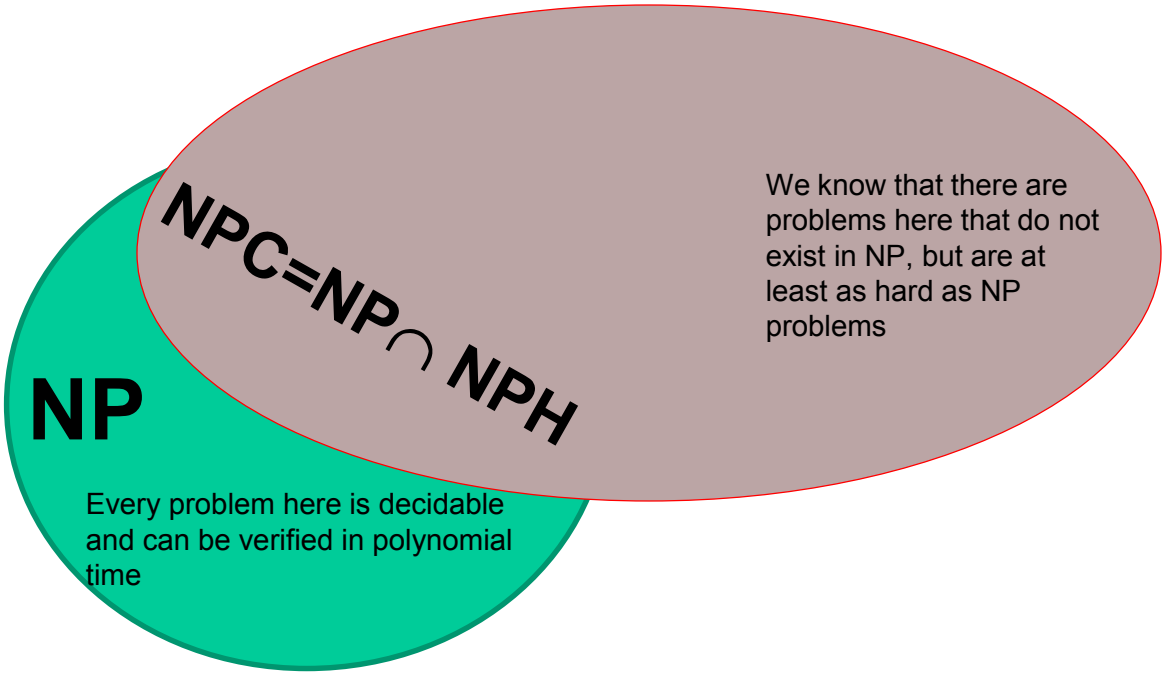
There are certain NP-Hard problems that also exist in NP. They are decidable, verifiable in polynomial time and are a polynomial reduction of an NP problem.

These are said to be **NP-Complete**.

Restated, these NP-Complete problems are the intersection of NP and NP-Hard problems

In our diagram, NPC=NP$\cap$NPH

# NP-Complete

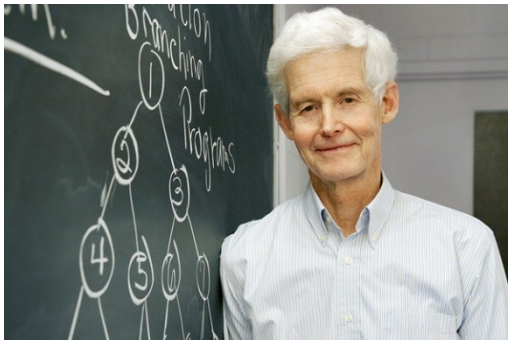

NPC=NP∩ NPH
We know that there are problems here that do not exist in NP, but are at least as hard as NP problems
NP
Every problem here is decidable and can be verified in polynomial time

# NP-Complete: Properties and Implications

- They are all decidable (since they are also in NP)
- Since they are also in NP, then any one can be related to another through polynomial reduction (Cook's Theorem)

# Cook's Theorem

Any NP-complete problem, using a polynomial-time function, can be reduced to SAT.



Stephen Cook

Consequence: If ***any*** NP-complete problem can be shown to be in P, then ***all*** NP-complete problems are in P

# SAT trivia

The canonical form for SAT problems is 3-SAT. This form is called the Conjunctive Normal Form (CNF) (AND'ing of clauses of OR's) In 3-SAT there are exactly 3 literals in each disjunctive clause

$$
\begin{array}{lllllllll}
( & a_1 & \vee & b_1 & \vee & c_1 & ) & \wedge \\
( & a_2 & \vee & b_2 & \vee & c_2 & ) & \wedge \\
( & a_3 & \vee & b_3 & \vee & c_3 & ) & \wedge \\
. & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . \\
( & a_n & \vee & b_n & \vee & c_n & ) &
\end{array}
$$

Any SAT expression can be put into 3-SAT CNF in polynomial time using the rules of Boolean logic

Actually, Cook's theorem more properly applies to 3-SAT, but 3-SAT is only a polynomial away from unrestricted SAT

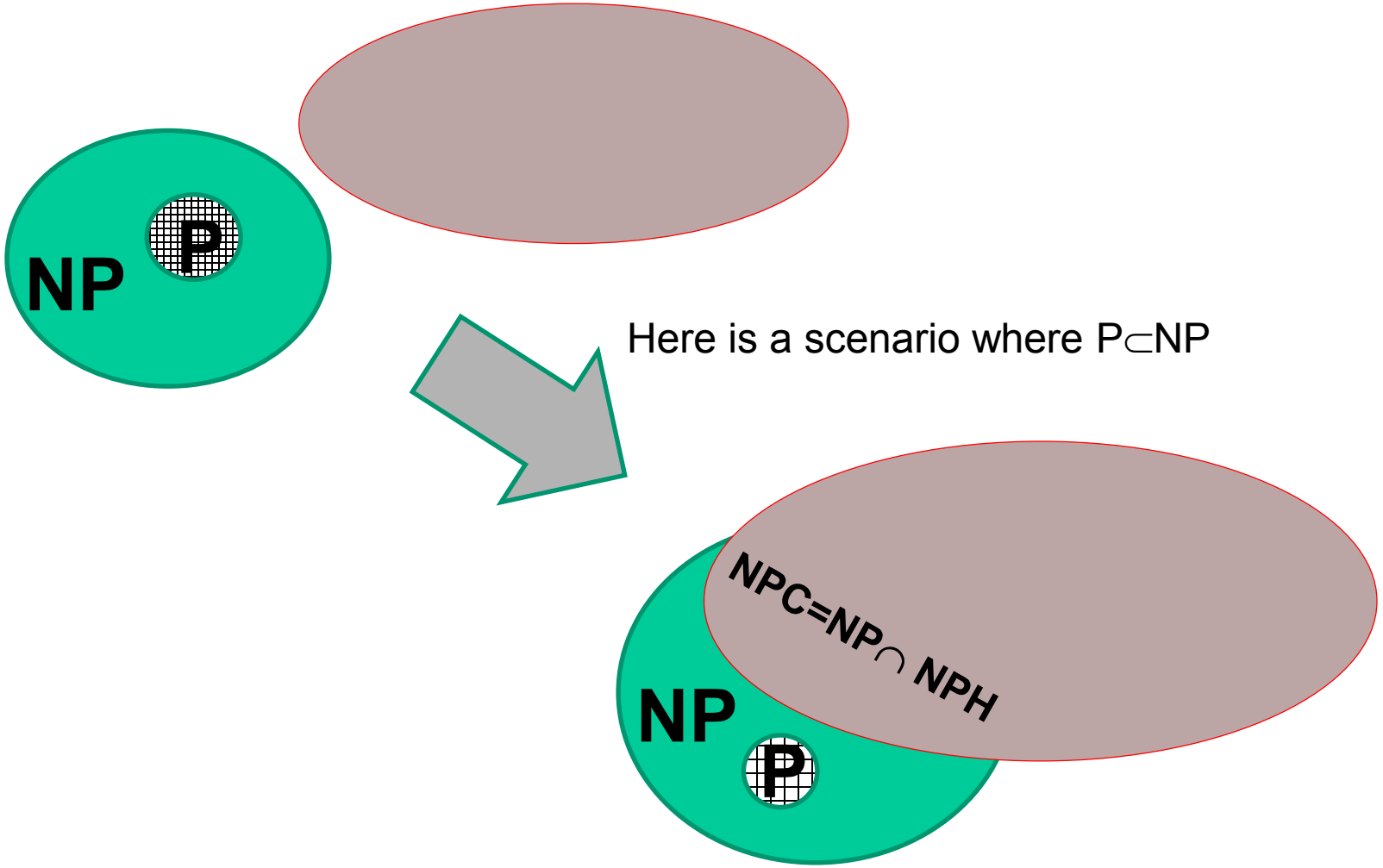
# NP-Complete problems

So, what are these NP Complete problems (problems which can be mapped into SAT or any other NP-Complete problem in polynomial time)?

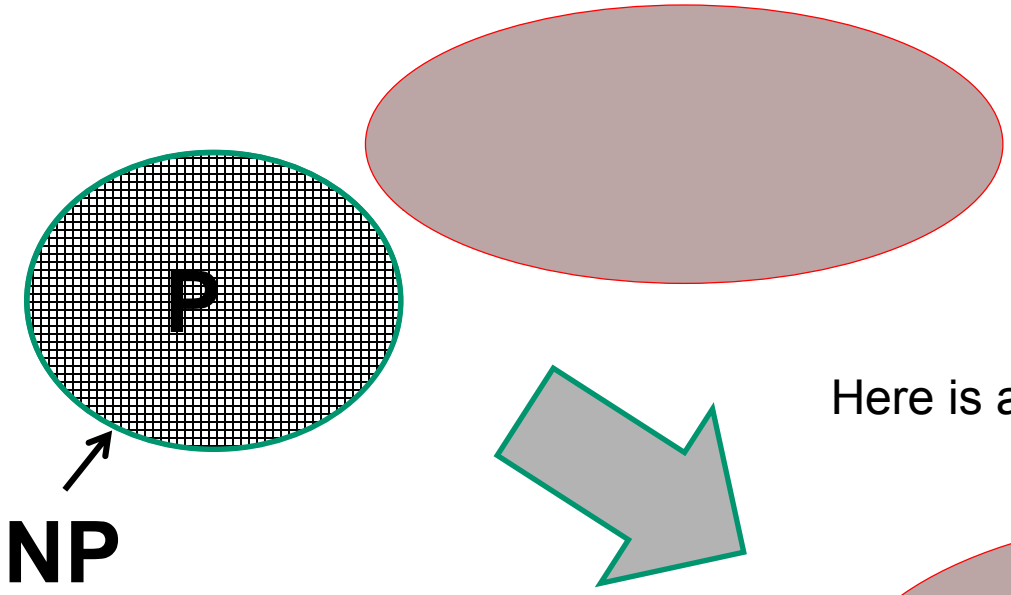
Here are a few:

- Map Coloring Problem
- Traveling Salesman Problem (TSP)
- Bin Packing Problem
- Knapsack problem
- Hamiltonian Graph Problem (generalization of TSP)
- Integer Factorization
- Smallest Superstring

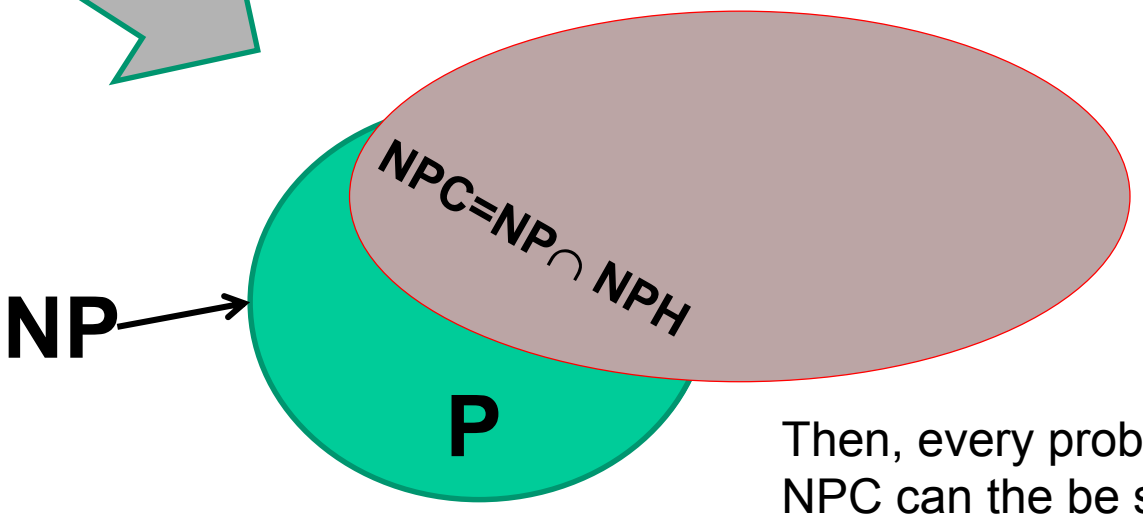
# Don’t forget about P!



# But what if P=NP……?




Here is a scenario where P=NP




Then, every problem in NPC can the be solved in polynomial time!

# Implications

We are still not sure if P=NP. In fact, we are pretty sure is does not, but have no proof either way.

BUT…

Suppose P=NP. Then, by Cook's Theorem, not only is it possible to have an efficient solution to a problem in NP, it is possible to have an efficient solution to ***every*** problem in NP!

# Another word about Cook's Theorem

If P=NP can be proven, and if the algorithm can be found, then the world as we know it changes!

Consider encryption. The most robust encryption scheme available is the **R**ivest-**S**hamir-**A**delman algorithm. The RSA algorithm creates a public key using the product of two very large prime numbers. The sender knows the factors, the hacker sees only the product. The extreme difficulty for the hacker to factor this product (Integer Factorization is NP-complete) is the basis upon which secure messaging is possible. If the public key can be factored, the security evaporates.

# NP-complete problems and reality

So, what are we going to do when confronted with an NP-complete problem?

- If the instance is of small size, we may solve it
- We may seek an heuristic solution

′ευριστικέιν→′ευριστικος

*to find* → *inventive*

# HEURISTICS

- An heuristic is an easy way to get an ‘answer’ to a hard problem. In return, a price is paid:
  - The solution may not be the best (*e.g.* greedy algorithm)
  - It may not deliver all possible solutions
  - The solution may be an approximation (*e.g.* simulated annealing)

- Sometimes heuristics solve sub-problems, or easy analogs that are not quite the same as the ‘real’ problem but can be easily solved

# GreedyAlgorithm
# Bin Packing

- First Fit (greedy)
- First fit descending (greedy but better)
- Full bin (greedy but better)

BIN PACKING HEURISTIC: FIRST FIT ALGORITHM

| | | bin1 | b2 | b3 | b4 | b5 | b6 |
|---|---|---|---|---|---|---|---|
| 17 | | 17 | 9 | 27 | 5 | 15 | 30 |
| 22 | | 22 | 35 | 19 | 31 | 25 | |
| 8 | | 8 | 2 | | 11 | | |
| 9 | | 3 | | | | | |
| 35 | | | | | | | |
| 27 | | | | | | | |
| 2 | | | | | | | |
| 19 | | | | | | | |
| 5 | | 50 | 46 | 46 | 47 | 40 | 30 |
| 31 | | | 4 | 4 | 3 | 10 | |
| 15 | | | | | | | |
| 25 | | | | | | | |
| 11 | | | | | | | |
| 30 | | | | | | | |
| 3 | | | | | | | |
| 5.18 | Lower Bound | | | | | | |

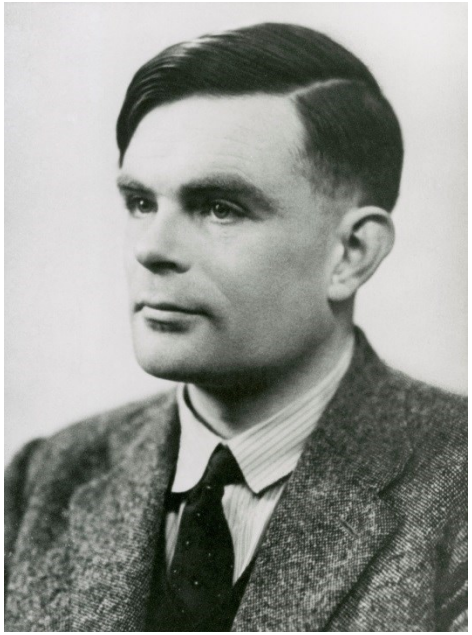
FIRST FIT DESCENDING ALGORITHM

| | | bin1 | b2 | b3 | b4 | b5 | b6 |
|---|---|---|---|---|---|---|---|
| 35 | | 35 | 31 | 30 | 27 | 25 | 8 |
| 31 | | 15 | 19 | 17 | 22 | 11 | 2 |
| 30 | | | | 3 | | 9 | |
| 27 | | | | | | 5 | |
| 25 | | | | | | | |
| 22 | | | | | | | |
| 19 | | | | | | | |
| 17 | | | | | | | |
| 15 | | | | | | | |
| 11 | | 50 | 50 | 50 | 49 | 50 | 10 |
| 9 | | 0 | 0 | 0 | 1 | 0 | 40 |
| 8 | | | | | | | |
| 5 | | | | | | | |
| 3 | | | | | | | |
| 2 | | | | | | | |

# Heros

John Von Neuman

Noam Chomsky



Alan Turing