P, NP, and Complexity

- Six fundamental facts
- One rule of thumb
- Three fundamental notions
- One fundamental question

FACULTY OF
ENGINEERING &
INFORMATION
TECHNOLOGIES

Peter Eades
School of IT

THE UNIVERSITY OF
SYDNEY

Fundamental Fact #1

Exponential functions are eventually bigger than polynomial functions

Fundamental Fact #1

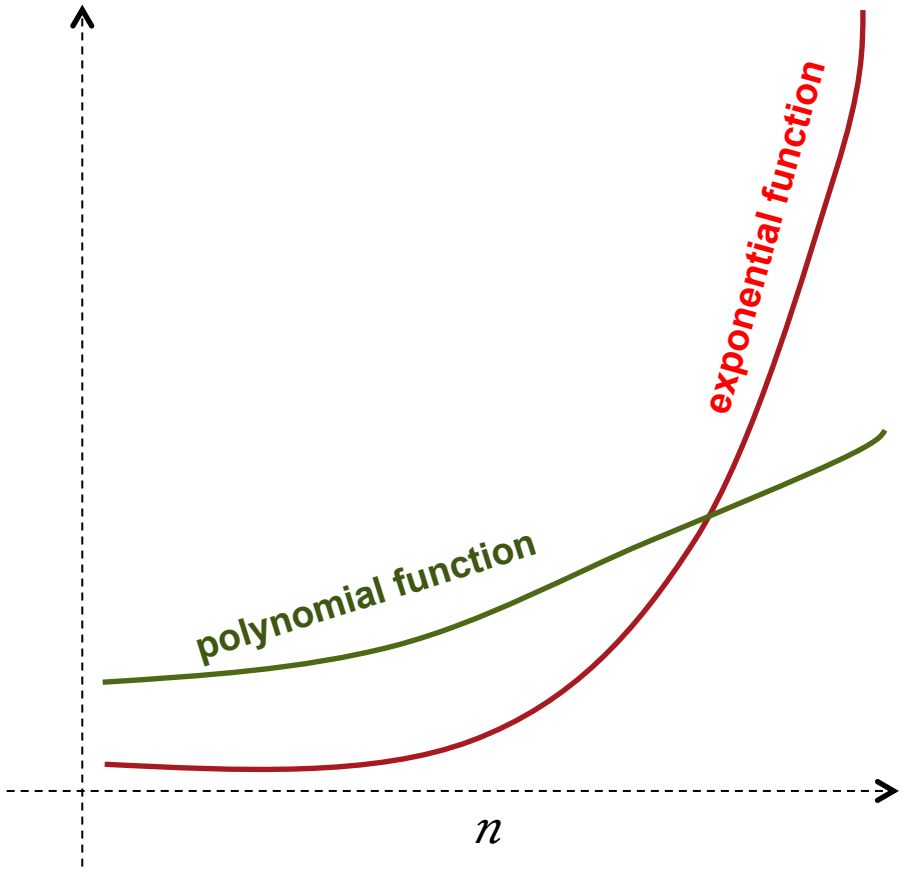
Exponential functions are eventually bigger than polynomial functions

Examples of exponential functions:

- $f(n) = 2^n$
- $f(n) = e^{n-7} + 17$
- $f(n) = n^{2n-1}$

Examples of polynomial functions:

- $f(n) = n^2 + n + 21$
- $f(n) = 26n^7 + 4n^5 + 231n^2 + 21$



Fundamental Fact #2

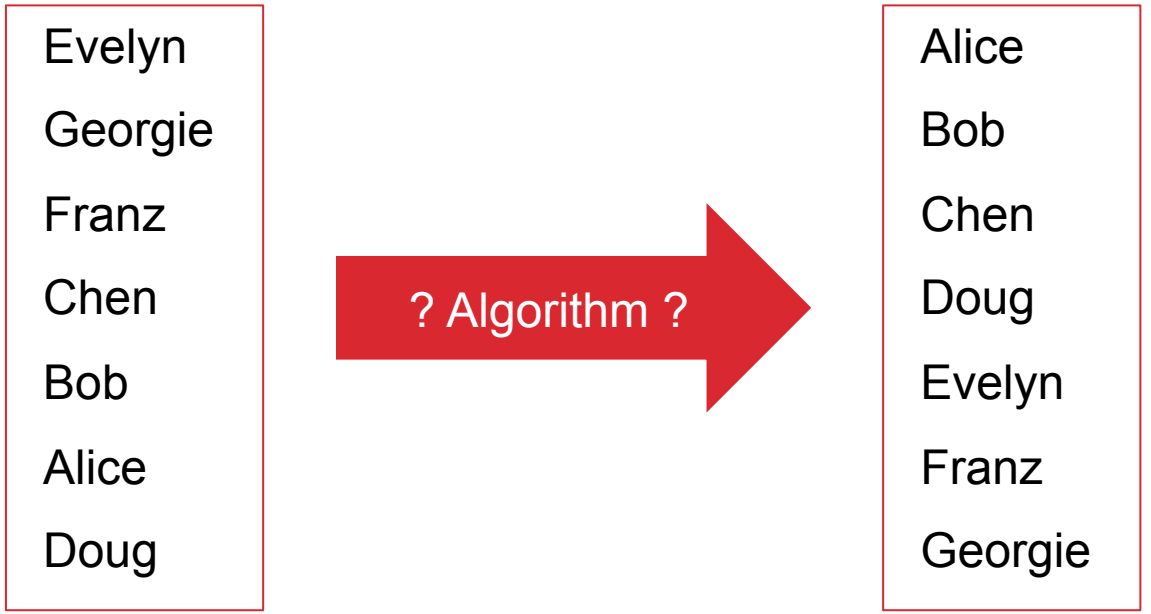
Fundamental Fact #2

Some algorithms are efficient, some are not

Fundamental Fact #2

Some algorithms are efficient, some are not

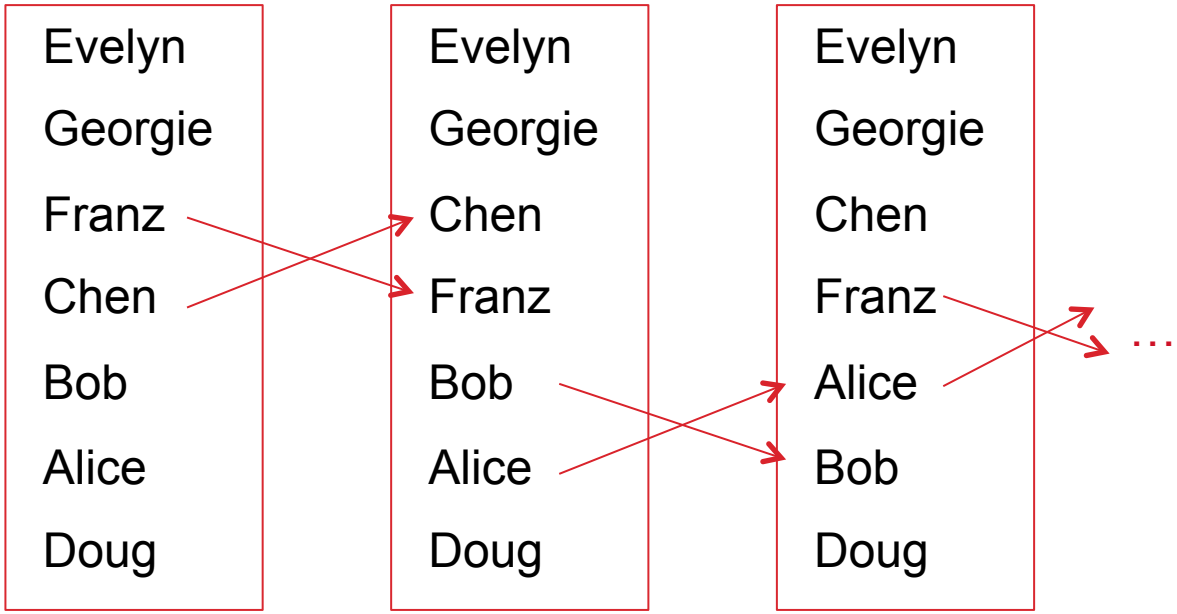
Sorting problem:



There are many different algorithms for sorting

Basic step:

› If two things are out of order, we can swap them to improve the *sortedness* of the list



exhaustingSort

- Exhaustively swap to get every possible ordering of the list
- At each step, check to see if the list of ordered

Run-time for ***exhaustingSort***:

- You need to consider $n!$ different orderings of the list
- This takes time at least proportional to $n! \cong \sqrt{2\pi n}\,(n/e)^{\uparrow n}$
- This is infeasible

bubbleSort

- Scan the list from top to bottom
- If the ith and $(i+1)$th elements are out of order, then swap them
- Keep scanning and swapping until the list is sorted

Run-time for ***bubbleSort***:

- you need to scan the list n times
- make $0.5n^2 - 1.5n + 2$ swaps.

This takes time proportional to n^2 .

geeky_bubbleSort

- Same old ***bubbleSort*** algorithm, but use really clever coding in *assembler* and *C* languages to make it go faster.
- Run on a fast computer.

Run-time for ***geeky_bubbleSort***:

- you still need to scan the list n times
- you still make $0.5n^2 - 1.5n + 2$ swaps.

It still takes time proportional to n^2 .

But it seems to be much faster … … …

AISASort

- Measure the "sortedness" of the sequence:

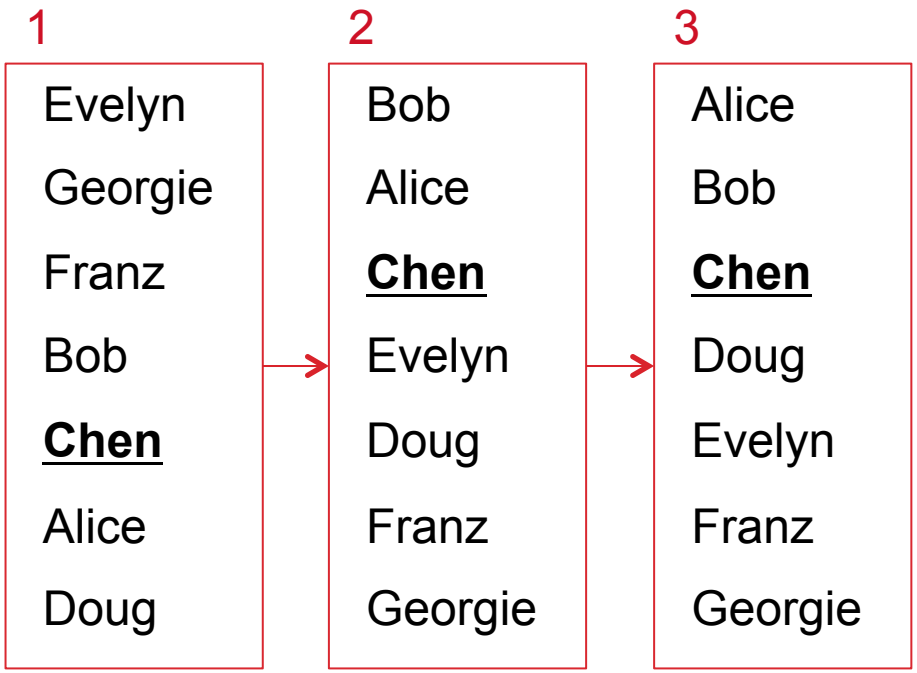
$$sortedness(x\downarrow 1, x\downarrow 2, \ldots x\downarrow n) = \sum i=1\uparrow n-1 \square (x\downarrow i+1 - x\downarrow i)$$

- Choose swaps that increase *sortedness* as much as possible
- Use an AI technique called "simulated annealing" to search for a maximally sorted sequence

Run-time for ***AISASort***: exponential

quickSort

1. Choose an element, called a *pivot*, from the list.
2. Swap things around so that smaller things come before the pivot, and larger things come after it
3. Recursively sort the sub-list of lesser elements and the sub-list of greater elements.



Run-time for ***quickSort***: proportional to $n \log n$

How long does it take to sort n things?

n	exhaustingSort	bubbleSort	geeky_bubbleSort	AISASort	quickSort
10	About 100	0	0	0.001	0
100	-	0	0	4.924*	0
10000	-	0.123	0.170	12266.808*	0.006
1000000	-	1296.560	734.422	-	0.111

Remarks:

- ***quickSort*** is excellent
- ***bubbleSort*** and ***geeky_bubbleSort*** are feasible
- ***AISASort*** is infeasible (as well as ***exhaustingSort***)

Rule of Thumb #1

Rule of Thumb #1

An algorithm that runs in exponential time is not feasible; an algorithm that runs in polynomial time may be feasible.

THE UNIVERSITY OF SYDNEY

Rule of Thumb #1

An algorithm that runs in exponential time is not feasible; an algorithm that runs in polynomial time may be feasible.

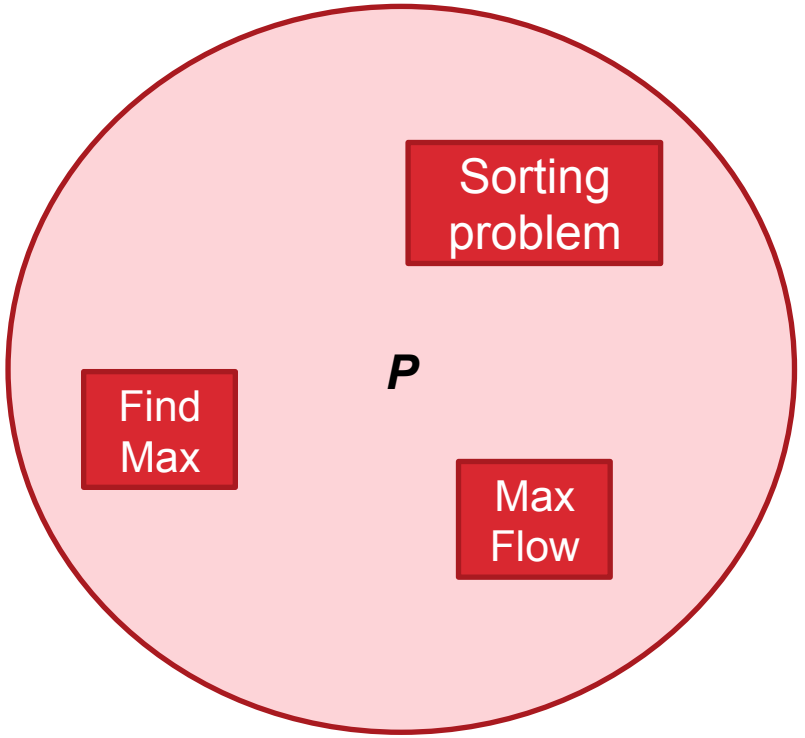
- Infeasible
  - exhaustingSort
  - AISASort
- Feasible
  - bubbleSort
  - quickSort

Fundamental Notion #1: P

Fundamental Notion #1: P

P is the set of all problems that can be solved in polynomial time

Fundamental Notion #1: P



Fundamental Fact #3

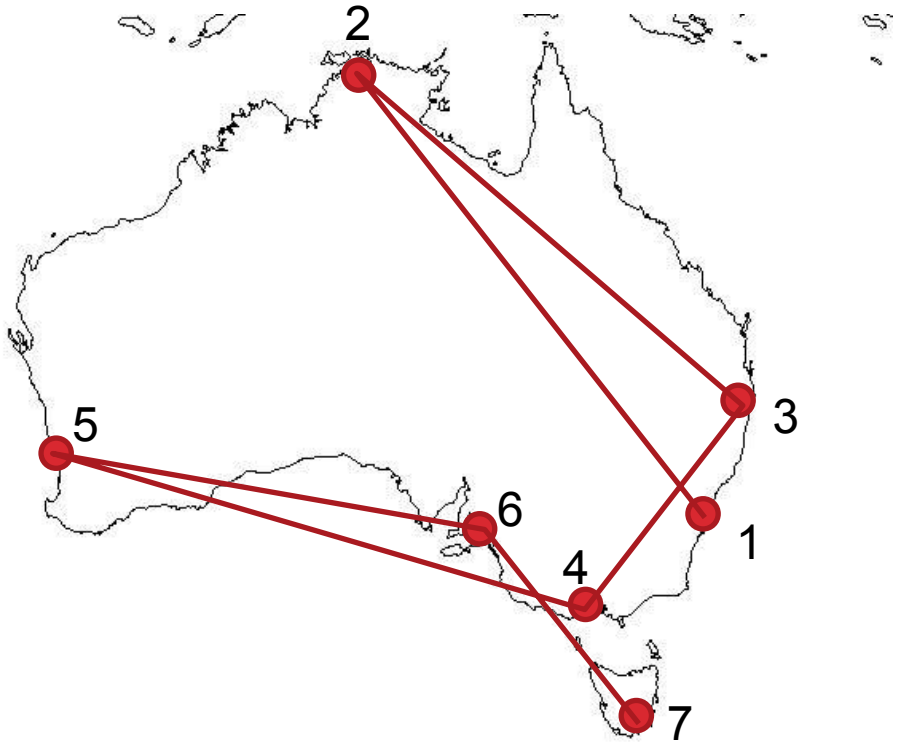
Fundamental Fact #3

Some problems can be solved with efficient algorithms, and some others … maybe not

Some problems can be solved with efficient algorithms, and some others … maybe not

Travelling salesman problem:

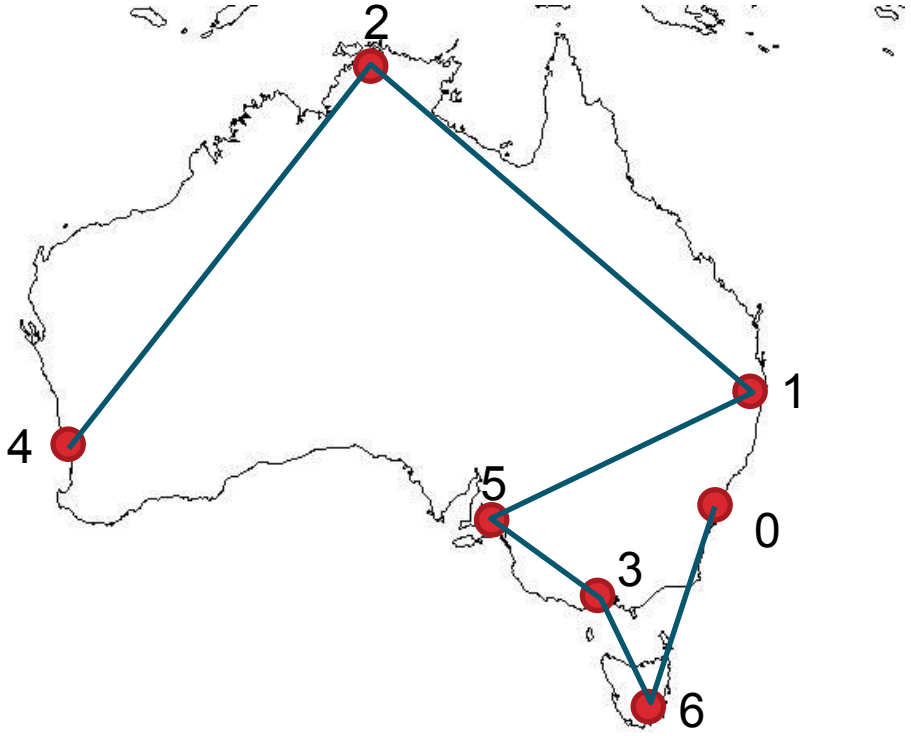
- How can we choose a route around n cities to minimise the total distance travelled?
- We need to order the cities appropriately



Some problems can be solved with efficient algorithms, and some others … maybe not

Travelling salesman problem:

- How can we choose a route around n cities to minimise the total distance travelled?
- We need to order the cities appropriately



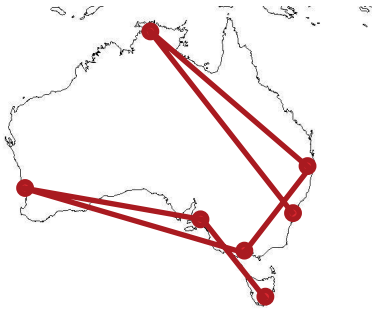
There are many different algorithms for the traveling salesman problem

Basic problem:

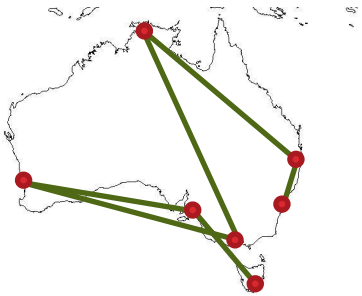
› We need to swap around the order of cities to decrease the length travelled

Like the sorting problem

1. Sydney
2. Darwin
3. Brisbane
4. Melbourne
5. Perth
6. Adelaide
7. Hobart



1. Sydney
2. Brisbane
3. Darwin
4. Melbourne
5. Perth
6. Adelaide
7. Hobart



exhaustingTSP

- Exhaustively swap to get every possible ordering of the cities
- At each step, compute the total distance travelled
- Keep the tour with the minimum total distance

Run-time for ***exhaustingTSP***:

- You need to consider $n!$ different orderings of the cities
- This takes time at least proportional to $n! \cong \sqrt{2\pi n}\,(n/e)^{\uparrow n}$
- This is infeasible

bubbleTSP

- Scan the current list of cities
- If swapping the ith and $(i+1)$th cities would decrease distance, then swap them
- Keep scanning and swapping until no swap decreases distance

Run-time for ***bubbleTSP***:

- Hard theorem: it can take exponentially time, but not as bad as ***exhaustingTSP***

Effectiveness of ***bubbleTSP***:

- Does not give optimal results
- Gives mediocre results

AISA_TSP

- Choose swaps that decrease distance as much as possible
- Use an AI technique called “simulated annealing” to search for an ordering with smallest total distance

Run-time for ***AISA_TSP***:

- Hard theorem: it can take exponentially long
- Faster than ***exhaustingTSP***
- Slower than ***bubbleTSP***

Effectiveness of ***AISA_TSP***:

- Does not give optimal results
- Gives mediocre results, but better than ***bubbleTSP***

How long does it take to find a solution to TSP?

n	AISA_TSP
10	0.052
100	96.013
10000	-
1000000	-

Remarks:

- ***We don't know any algorithm for TSP that runs fast and gives an optimal result***

Fundamental Fact #3

Fundamental Fact #3

Some problems can be solved with efficient algorithms, and some others … maybe not

(continued)

Clique problem:

- Given a network N of size n and an integer k
- Does N have k nodes that are all connected to each other?

Some people:

- Alice, Andrea, Annie, Amelia, Bob, Brian, Bernard, Boyle

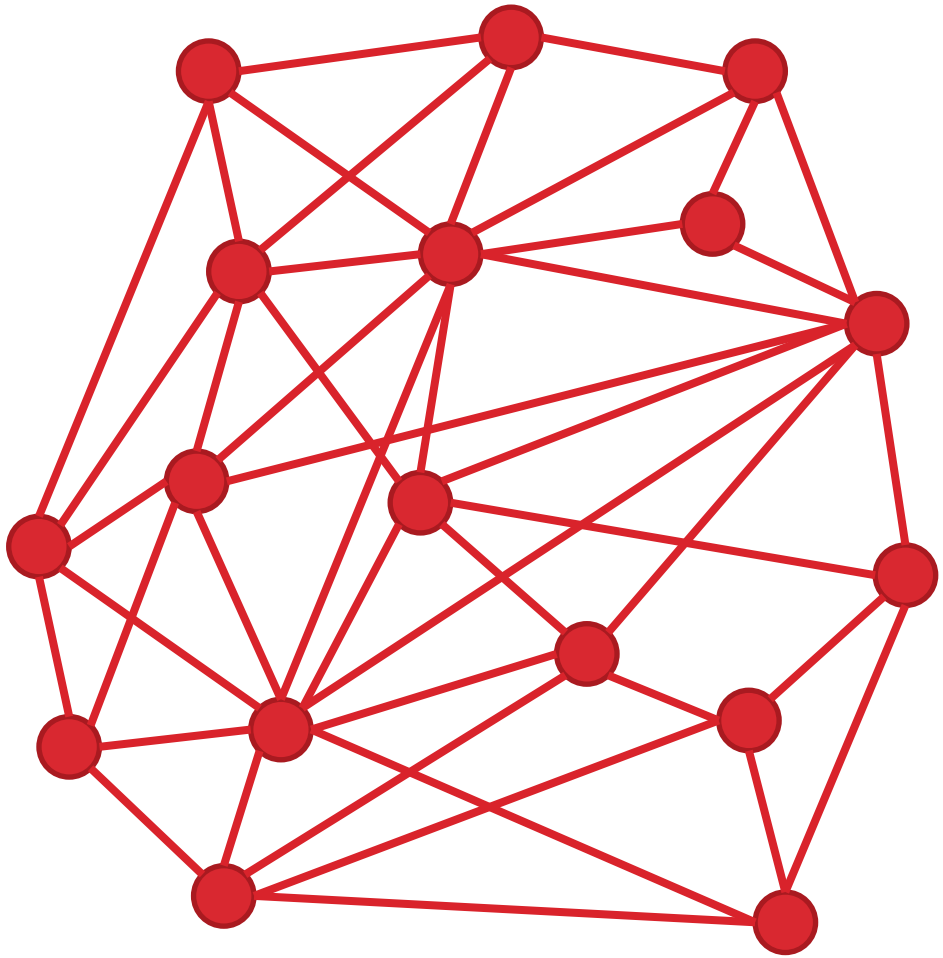
Some connections

- Bob is connected to Alice
- Bob is connected to Andrea
- Bob is connected to Amelia
- Brian is connected to Alice
- Brian is connected to Andrea
- Brian is connected to Amelia
- Boyle is connected to Alice
- Boyle is connected to Andrea
- Boyle is connected to Annie
- Bernard is connected to Alice
- Bernard is connected to Andrea
- Bernard is connected to Annie

Is there a clique of size 3 among these people?

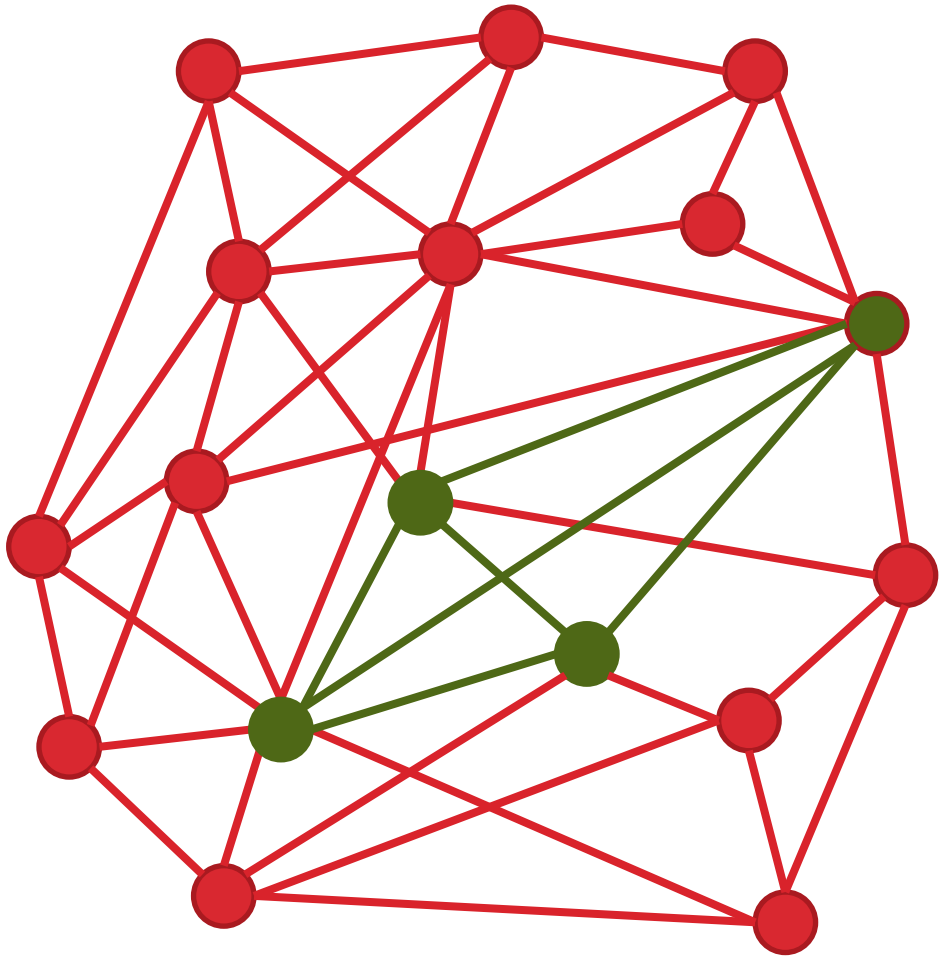
(Clique: a group of people who are all connected to each other)

Another network



Is there a clique of size 5 among these nodes?

cliques



Clique of size 4

exhaustingClique

- Test every subset of k nodes, check to see if these k nodes form a clique

Run-time for ***exhaustingClique***:

- You need to consider $(\blacksquare nk)$ different subsets of nodes
- If $k \cong n/2$ then is $(\blacksquare nk)$ exponential
- This is infeasible

Fundamental Fact #3

Fundamental Fact #3

Some problems can be solved with efficient algorithms, and some others … maybe not

(still continued)

The Satisfiabilty (SAT) problem

A statement P

If Peter is Australian, then Patrick is Australian, and

Either Patrick is Australian, or Patrick is Irish, or Paul is English, and

If Catherine is English then either Paul is not English or Jane is French, and

If Jane is French and Patrick is not Australian, then Peter is English, and

If Peter is English then Peter is not Irish, and

If Paul is not English then either Catherine is Australian or Jane is French, and

….

Question:

- Is there any way that this statement could be true?

SAT

Given an abstract proposition

If $p\downarrow 1$ then $p\downarrow 2$, and

Either $p\downarrow 3$, or $p\downarrow 1$, or $p\downarrow 4$, and

If $p\downarrow 1$ then either $p\downarrow 2$ or $p\downarrow 1$, and

If $p\downarrow 1$ and $p\downarrow 4$, then not $p\downarrow 3$, and

If $p\downarrow 2$ then not $p\downarrow 1$, and

If $p\downarrow 3$ then either $p\downarrow 3$ or $p\downarrow 1$, and

….

Are there truth values for $p\downarrow 1$, $p\downarrow 2$, … such that this statement is true?

exhaustingSAT

- Test every combination of true and false values for $p\downarrow 1$, $p\downarrow 2$, $p\downarrow 3$, ... , check to see whether the whole statement is true

Run-time for ***exhaustingSAT***:

- You need to consider $2\uparrow n$ different combinations of true and false values for $p\downarrow 1$, $p\downarrow 2$, $p\downarrow 3$, ...
- This is infeasible

Some problems can be solved with efficient algorithms, and some others … maybe not

The Travelling Salesman Problem, the Clique Problem, and SAT all share some characteristics:

- They can be solved by exhausting algorithms that use exponential time
- No-one knows any algorithms that run fast and always solve these problems

Remarks

- These problems are commercially important, and a huge amount of research has been done on them.
- There are many algorithms for specific versions, and many algorithms that almost work
- There are many other problems with the same characteristics

Fundamental Fact #4

Fundamental Fact #4

Sometimes we can efficiently check whether an answer is correct, even if we can't efficiently find a correct answer

Sometimes we can efficiently check whether an answer is correct, even if we can't efficiently find a correct answer

› *Clique Problem*:

- Given a set of k nodes, one can quickly check whether these k nodes are all connected to each other.
- But it seems difficult to find the *right* set of k nodes.

› *SAT*:

- Given a truth value for each variable pi it is easy to check whether these truth values make the statement true.
- But it seems hard to find the *right* truth values

› *Traveling Salesman Problem*:

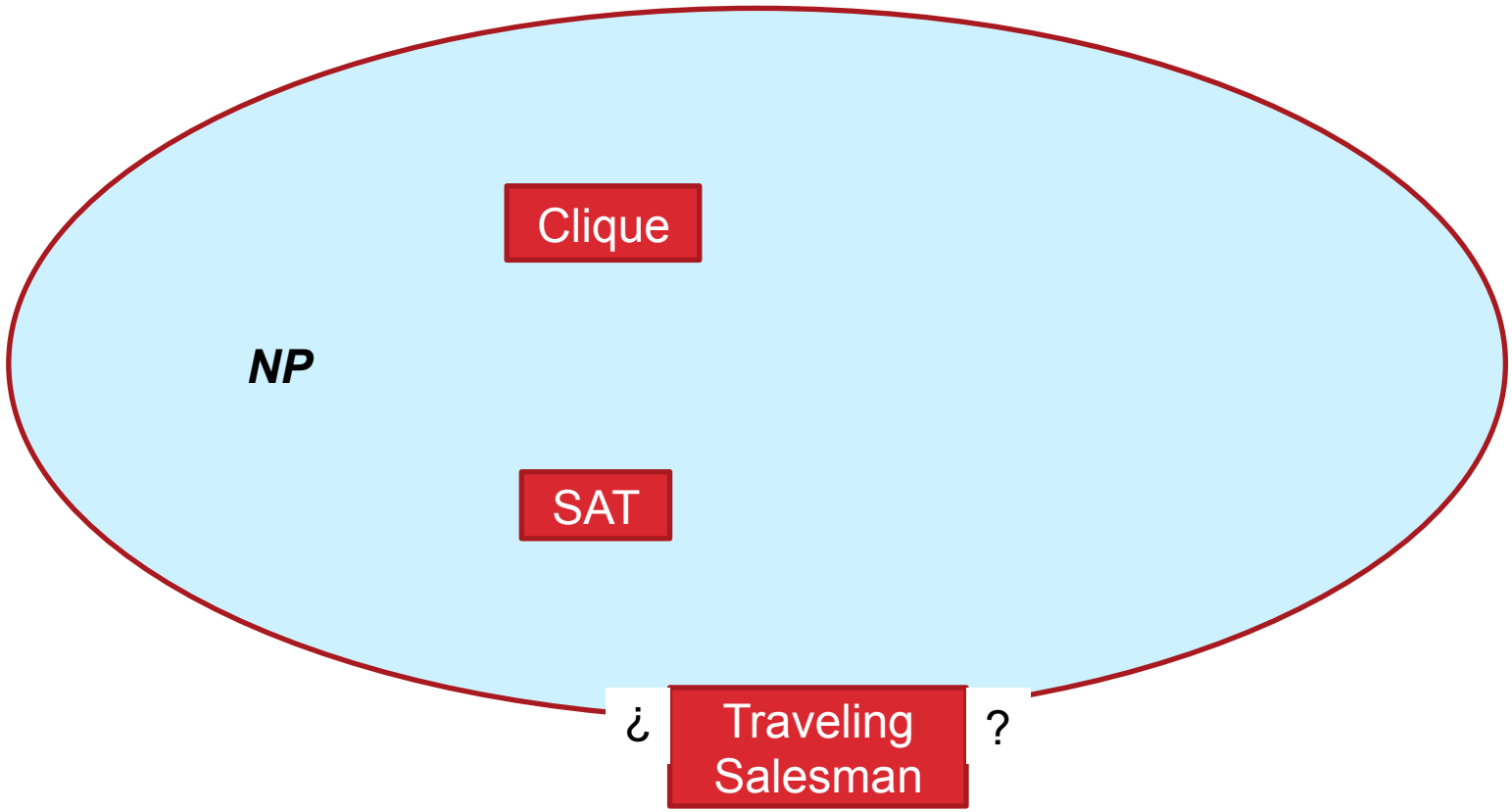
- Hmmmm … it may be hard even to check whether a given order of cities is optimal?

Fundamental Notion #2: NP

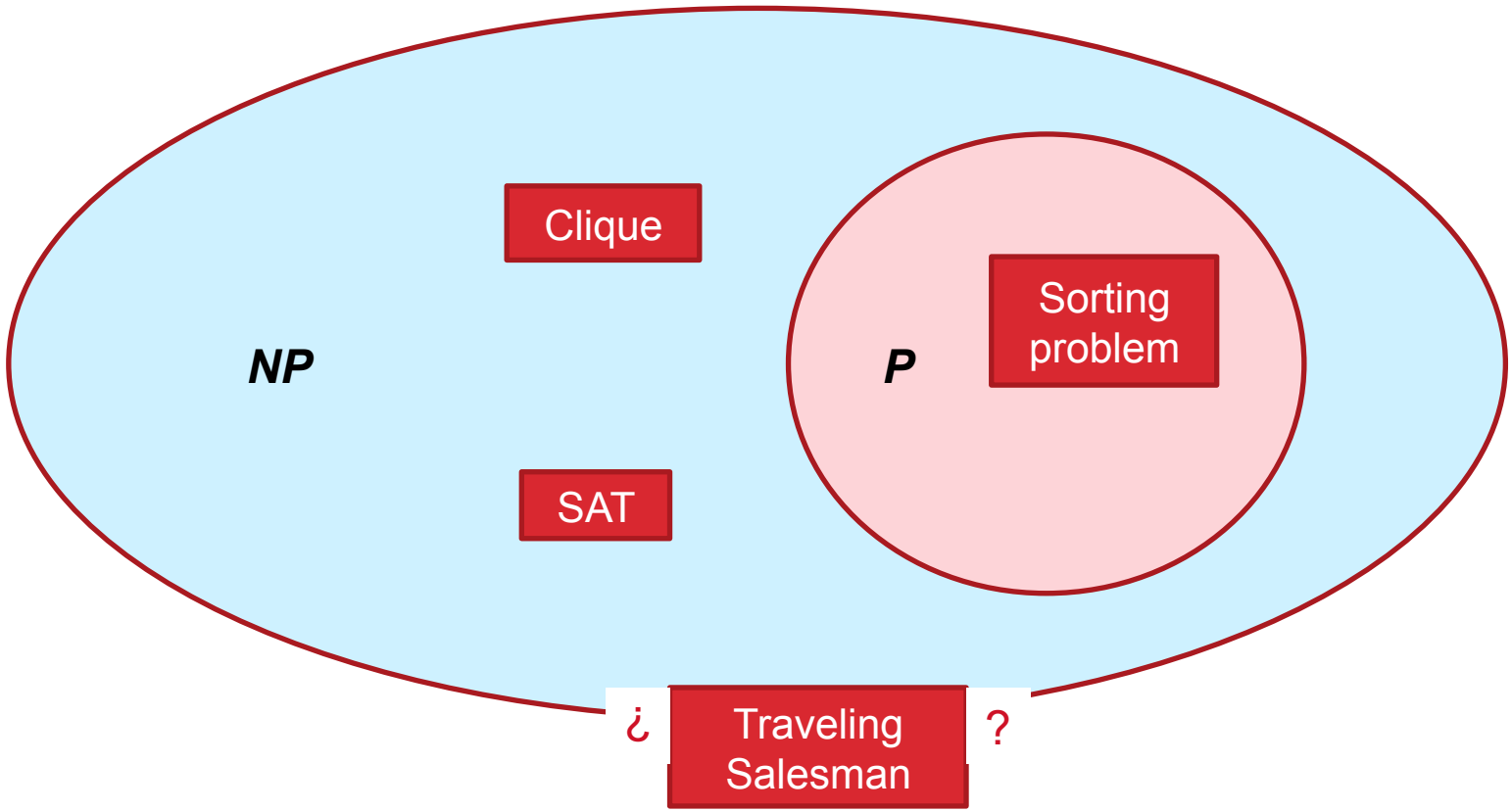
Fundamental Notion #2: NP

NP is the set of problems for which we can efficiently check to see whether a given answer is correct.

Fundamental Notion #2: NP



Fundamental Notion #2: NP



Fundamental Fact #5

Fundamental Fact #5

Some problems are harder than others

Some problems are harder than others

› The Clique problem is at least as difficult as SAT.

> Theorem
>
> If there were a good algorithm to solve the clique problem, then there would be a good algorithm to solve SAT.

Proof →

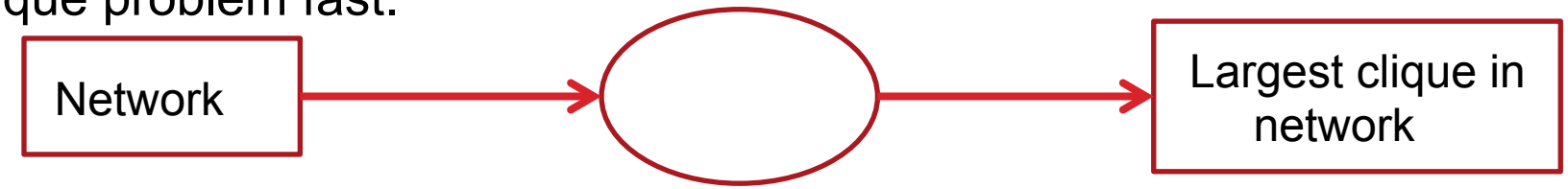
Clique is harder than SAT

Theorem

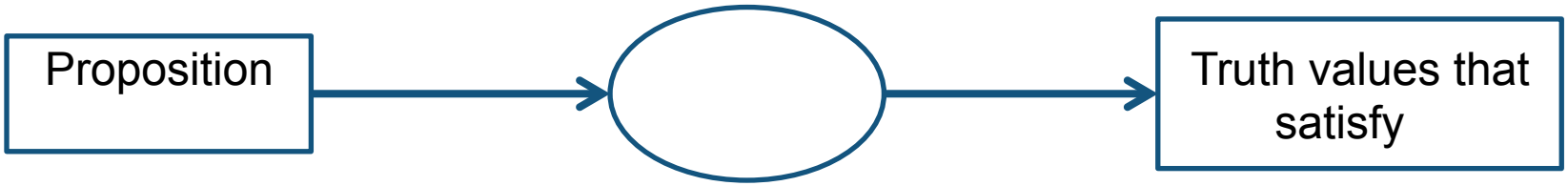
If there were a good algorithm to solve the clique problem, then there would be a good algorithm to solve SAT.

Proof:

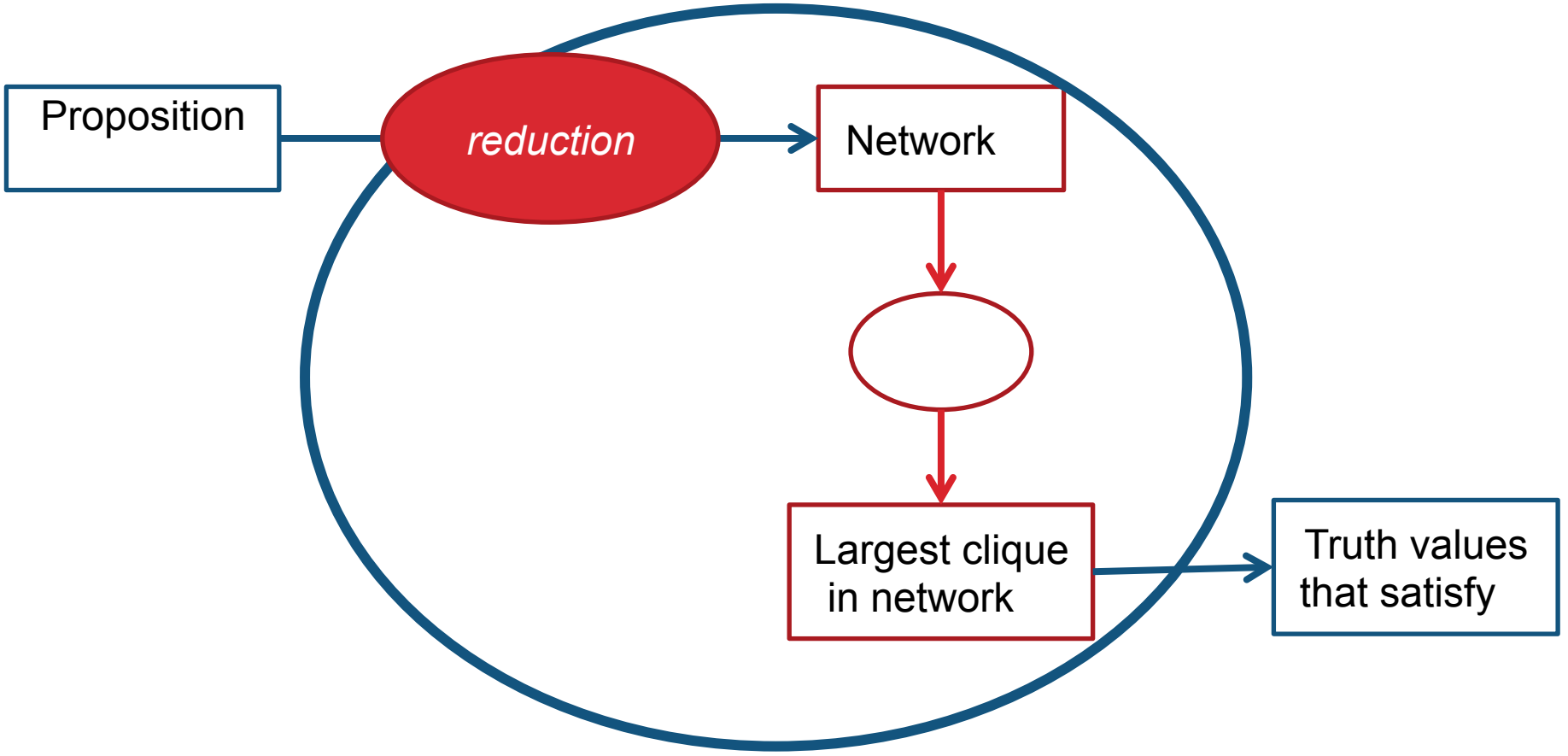
Suppose that we have a good fast algorithm *Aclique* that solves the clique problem fast.



Using *Aclique*,let's make an algorithm *ASAT* to solve SAT.



Clique is harder than SAT



Clique is harder than SAT

Either $p\downarrow 3$, or $not\ p\downarrow 1$, or $p\downarrow 4$, and

Either $p\downarrow 5$, or $not\ p\downarrow 2$, or $not\ p\downarrow 1$, and

Either $p\downarrow 6$, or $p\downarrow 2$, or $not\ p\downarrow 3$, and

Either $not\ p\downarrow 3$, or $p\downarrow 4$, or $p\downarrow 6$, and

Either $p\downarrow 5$, or $p\downarrow 2$, or $not\ p\downarrow 3$, and

Either $not\ p\downarrow 1$, or $not\ p\downarrow 4$, or

reduction

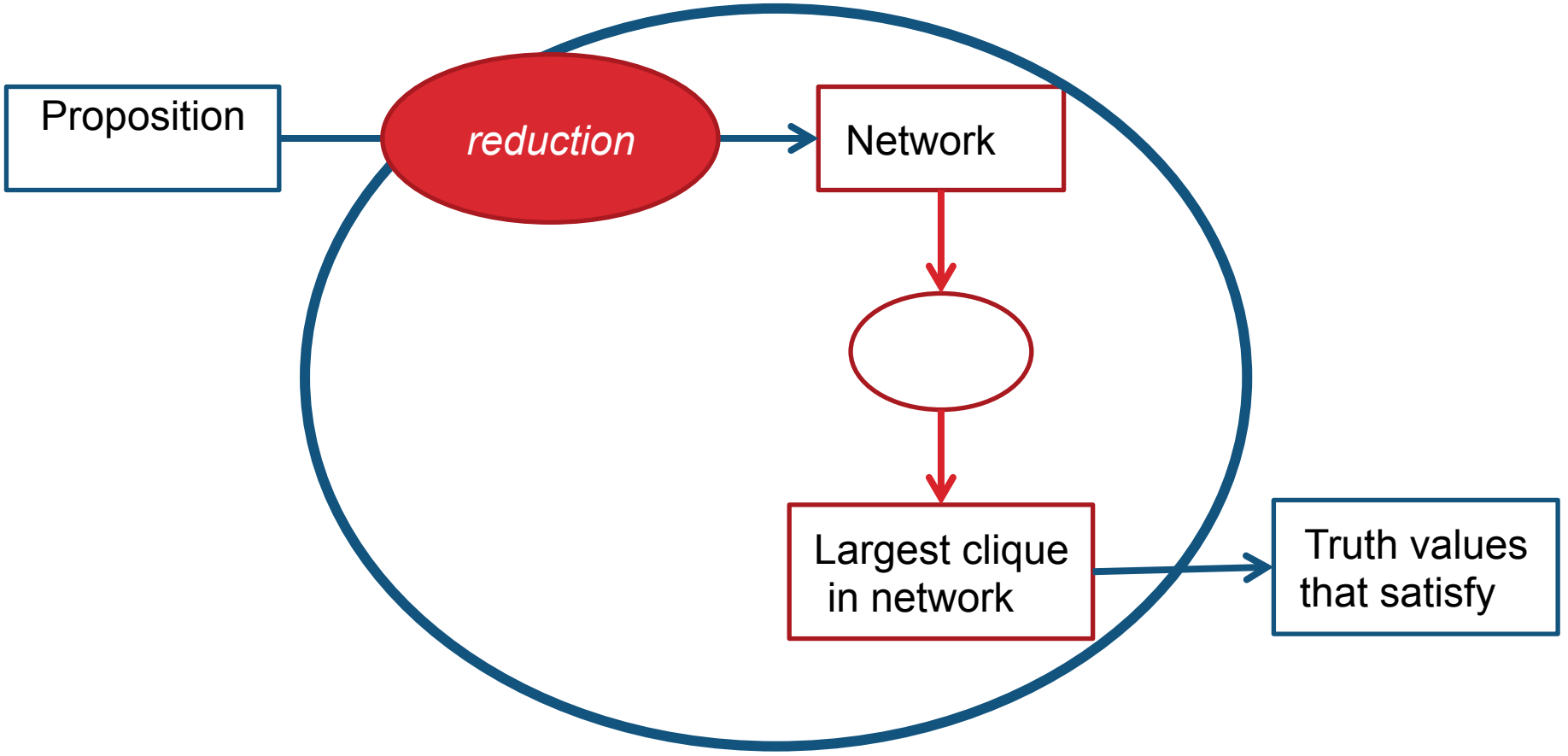
Nodes:

- For each i, there is a node for variable pi and a node for the negation $not\ pi$ of pi.

Connections:

- Connect two nodes if they do not occur in the same clause, and one is not the negation of the other.

Clique is harder than SAT



Clique is harder than SAT

Theorem

If there were a good algorithm to solve the clique problem, then there would be a good algorithm to solve SAT.

Proof:

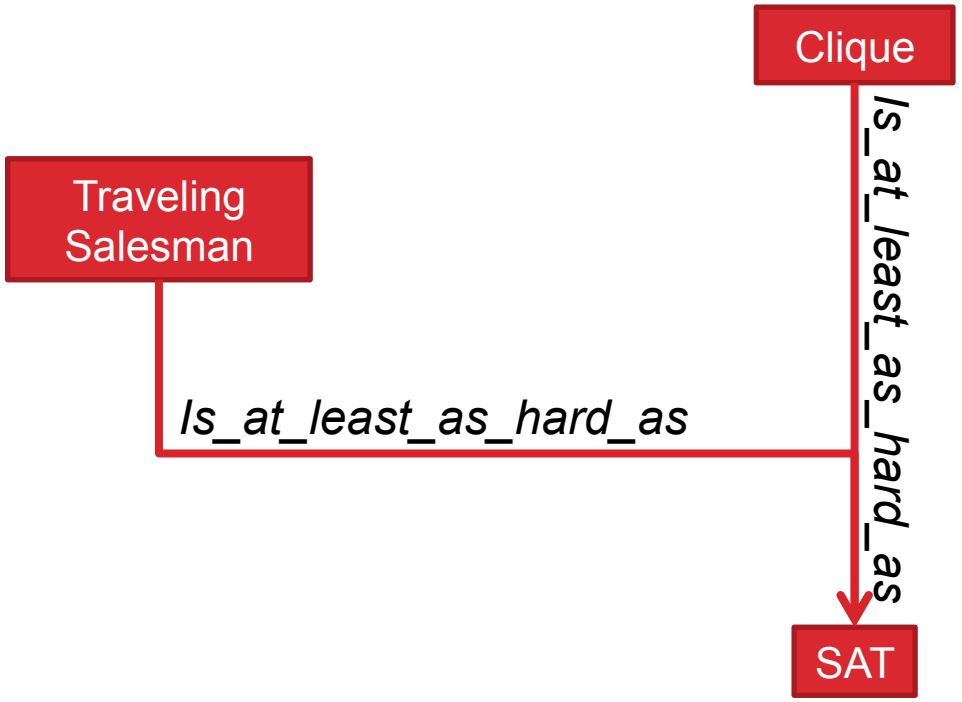
If we knew a good fast algorithm *Aclique* for the clique problem, then we could use it to make an algorithm *ASAT* to solve SAT.

- That is, if the clique problem were easy, then SAT would be easy
- That is, the clique problem is at least as difficult as SAT

Fundamental Fact #5

Some problems are harder than others

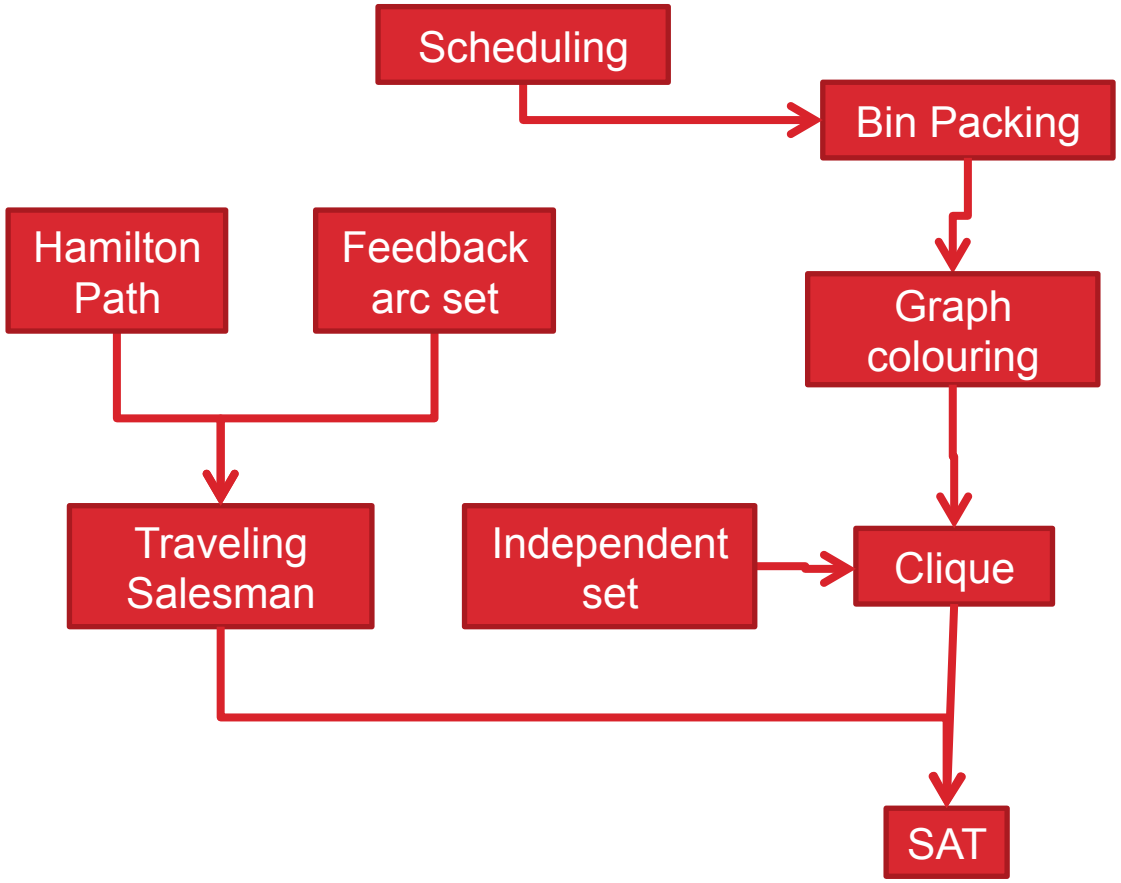
- Also, the travelling salesman problem is at least as hard as SAT.



Fundamental Fact #5

In fact, the relationship "***at_least_as_hard_as***" between problems has been very well investigated.

For many problems, there is a proof that this problem is at least as hard as SAT.



THE UNIVERSITY OF SYDNEY

Fundamental Fact #5

In fact, the relationship "***at_least_as_hard_as***" between problems has been very well investigated.

For many problems, there is a proof that this problem is harder than SAT.

Many problems are ***at_least_as_hard_as*** SAT.

- *Feedback Arc Set Problem*: solutions needed to detect deadlocks in communications systems
- *Scheduling problems*: solutions need in logistics industry, and in computer systems
- *Bin Packing problems*: solutions needed in manufacturing: steel industry, clothing industry
- *Many layout problems*: solutions needed for network visualization, newspaper layout, integrated circuit layout
- *Number theory problems*: solutions needed for cryptography

Fundamental Fact #6

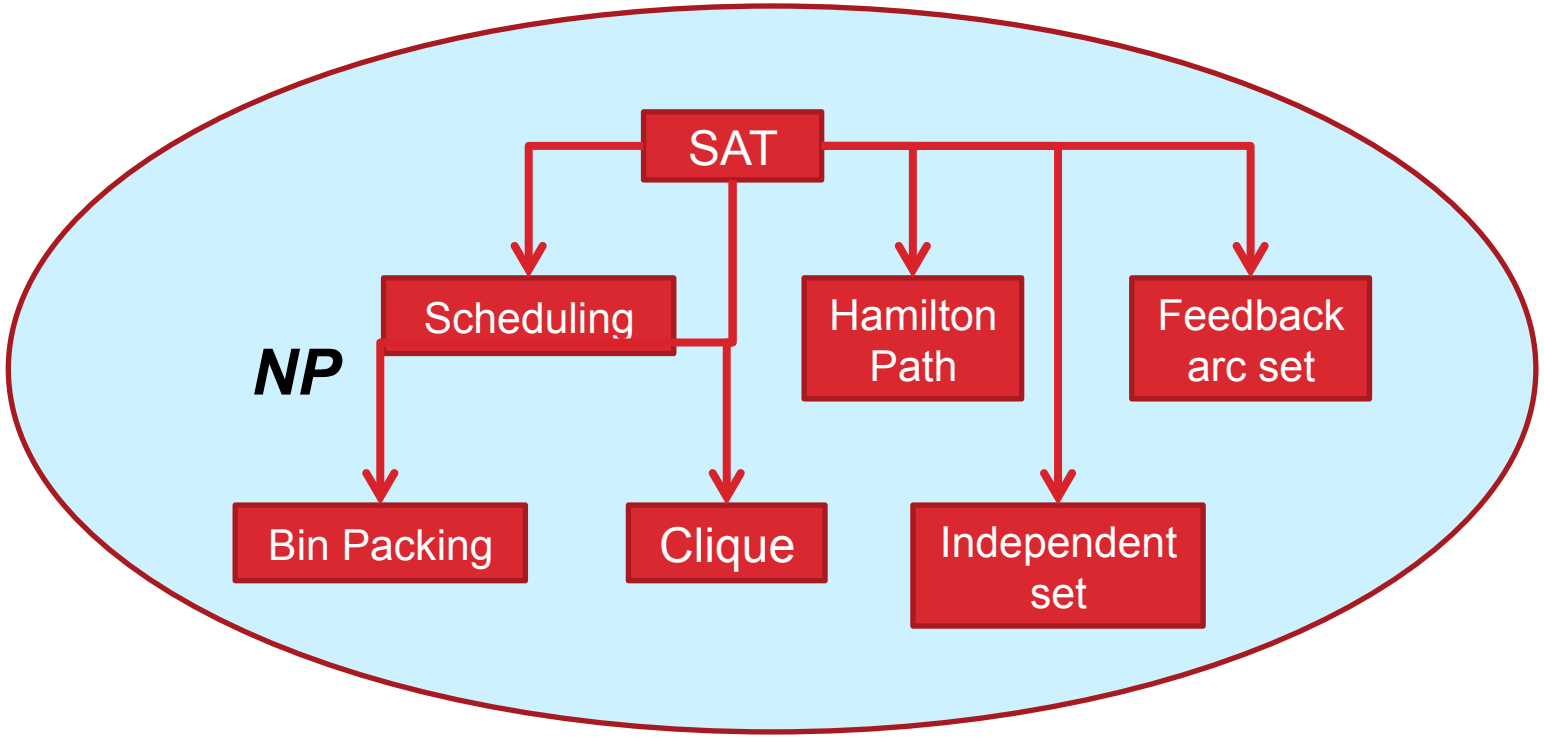
Fundamental Fact #6

SAT is at least as hard as every other problem in NP

(Cook's Theorem)

Fundamental Fact #6

SAT is *at_least_as_hard_as* every problem in NP



THE UNIVERSITY OF SYDNEY

Fundamental Fact #6

Cook's Theorem

SAT is "NP-complete"

Fundamental Notion #3: NP-complete

Fundamental Notion #3

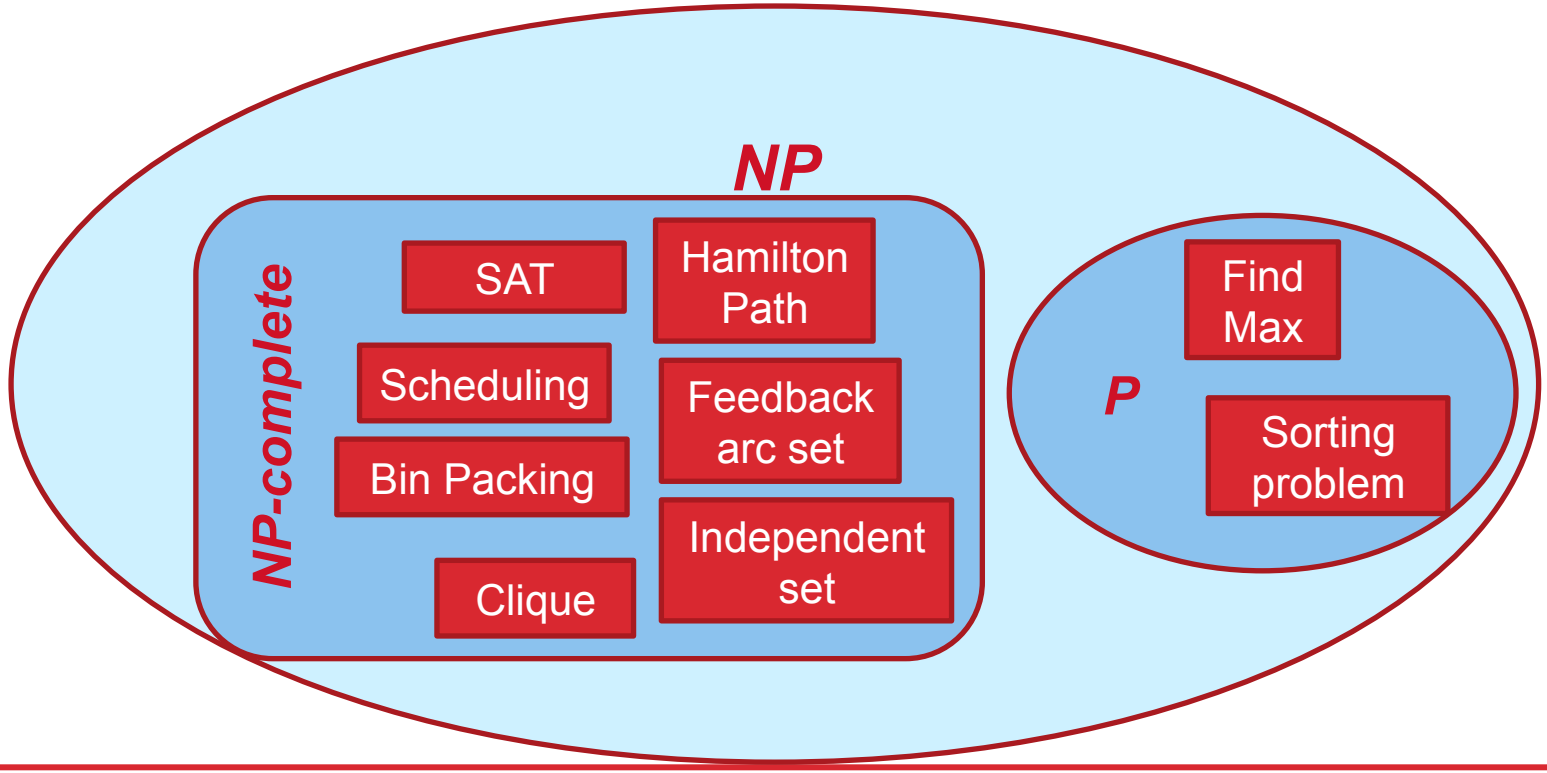
A problem is *NP-complete* if it is

- **In NP, and**
- **At least as difficult as every other problem in NP**

Fundamental Notion #3

A problem is *NP-complete* if it is

- **In NP, and**
- **At least as difficult as every other problem in NP**



Fundamental Fact #7

Fundamental Fact #7

Many real-world problems are NP-complete.

Fundamental Fact #7

Many real-world problems are ***NP-complete***.

- *Feedback Arc Set Problem*: solutions needed to detect deadlocks in communications systems
- *Scheduling problems*: solutions need in logistics industry, and in computer systems
- *Bin Packing problems*: solutions needed in manufacturing: steel industry, clothing industry
- *Many layout problems*: solutions needed for network visualization, newspaper layout, integrated circuit layout
- *Number theory problems*: solutions needed for cryptography

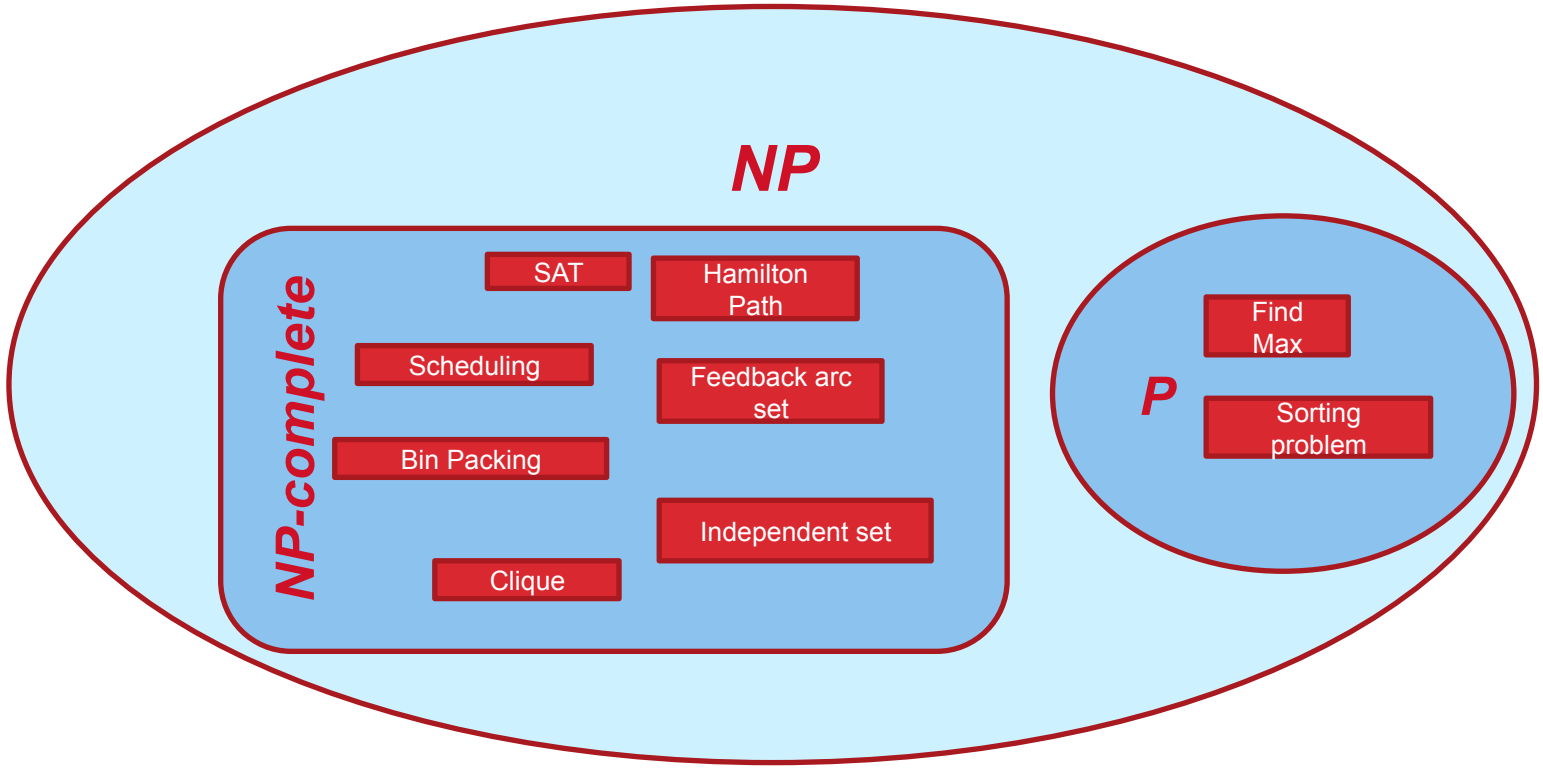
Fundamental Question #1

Fundamental Question #1

Does P equal NP?

Fundamental Question #1

P=NP ???



Fundamental Question #1

P=NP ???

- If P=NP then
  - All the real-world problems in NP have polynomial-time algorithms, and can be feasibly solved.
- If P≠ NP then
  - We must be satisfied with algorithms that do not work entirely.

Fundamental Question #1

P=NP ???

- To prove P=NP
  - You need to show that one NP-complete problem has a polynomial-time solution.
- To prove P≠ NP
  - You must show that one NP-complete problem does not have a polynomial-time solution

Fundamental Question #1

P=NP ???

- This is the most fundamental issue in Computer Science
- The problem is still unsolved

Fundamental Question #1

P=NP ???

› The investigation of this question and others like it is called “complexity theory”

The Fundamentals of P, NP, and Complexity

Fundamental Fact #1: ***Exponential functions are eventually bigger than polynomial functions***

Fundamental Fact #2: ***Some algorithms are efficient, some are not***

Rule of Thumb #1: ***An algorithm that runs in exponential time is not feasible; an algorithm that runs in polynomial time may be feasible.***

Fundamental Notion #1: ***P is the set of all problems that can be solved in polynomial time***

Fundamental Fact #3: ***Some problems can be solved with efficient algorithms, and some others … maybe not***

Fundamental Fact #4: ***Sometimes we can efficiently check whether an answer is correct, even if we can't efficiently find a correct answer***

Fundamental Notion #2: ***NP is the set of problems for which we can efficiently check to see whether a given answer is correct.***

Fundamental Fact #5: ***Some problems are harder than others***

Fundamental Fact #6: **SAT is at least as hard as every other problem in NP**

Fundamental Notion #3: **A problem is *NP-complete* if it is in NP, and at least as difficult as every other problem in NP**

Fundamental Fact #7: ***Many real-world problems are NP-complete***

Fundamental Question #1: ***Does P equal NP?***