Animals of the Complexity Zoo

Serge Kruk

October 31, 2022

Results of last course evaluations

- This chapter is not understood by students. Needs to be expanded.

$\mathcal{P} = \mathcal{NP}$ (A million dollar question.)

Consider the following pairs of problems:

- Given a graph with distances on the edges.

$\mathcal{P} = \mathcal{NP}$ (A million dollar question.)

Consider the following pairs of problems:

- Given a graph with distances on the edges.
  - Find the shortest path from a to b.

$\mathcal{P} = \mathcal{NP}$ (A million dollar question.)

Consider the following pairs of problems:

- Given a graph with distances on the edges.
    - Find the shortest path from a to b.
    - Find the longest path from a to b.

$\mathcal{P} = \mathcal{NP}$ (A million dollar question.)

Consider the following pairs of problems:

- Given a graph with distances on the edges.
  - Find the shortest path from a to b.
  - Find the longest path from a to b.
- Given a directed graph.

$\mathcal{P} = \mathcal{NP}$ (A million dollar question.)

Consider the following pairs of problems:

- Given a graph with distances on the edges.
    - Find the shortest path from a to b.
    - Find the longest path from a to b.
- Given a directed graph.
    - Find a tour that traverses each edge exactly once.

$\mathcal{P} = \mathcal{NP}$ (A million dollar question.)

Consider the following pairs of problems:

- Given a graph with distances on the edges.
  - Find the shortest path from a to b.
  - Find the longest path from a to b.
- Given a directed graph.
  - Find a tour that traverses each edge exactly once.
  - Find a tour that traverses each node exactly once.

$\mathcal{P} = \mathcal{NP}$ (A million dollar question.)

Consider the following pairs of problems:

- Given a graph with distances on the edges.
  - Find the shortest path from a to b.
  - Find the longest path from a to b.
- Given a directed graph.
  - Find a tour that traverses each edge exactly once.
  - Find a tour that traverses each node exactly once.
- As my kids used to say, "Do these sound alike?"

$\mathcal{P} = \mathcal{NP}$ (A million dollar question.)

Consider the following pairs of problems:

- Given a graph with distances on the edges.
  - Find the shortest path from a to b.
  - Find the longest path from a to b.
- Given a directed graph.
  - Find a tour that traverses each edge exactly once.
  - Find a tour that traverses each node exactly once.
- As my kids used to say, "Do these sound alike?"
  - Although seemingly related and superficially similar, the first of each pair is trivial; the second, probably not.

Why do we cover this in Algorithm design?

- As a computer scientist: Proving a problem to be NP-Complete is interesting

Why do we cover this in Algorithm design?

- As a computer scientist: Proving a problem to be NP-Complete is interesting
- As a software engineer: Knowing that a problem is NP-Complete means "try approximations"

Warning

- In this course, we will hand-wave the fine points of complexity classes. To really understand the material you need to take Theory of Computation.
    - Need precise definition of *language*.
    - Need precise definition of *Turing machine*.
    - Alternatively, need precise definition of *Lambda calculus*.
- Technically, only problems with Yes/No answers should be considered here but the conversion from an optimization problem to a Yes/No problem is trivial (assuming one knows how to binary search).

Technicality

- We will discuss only problems with binary answers (Yes/No) or (True/False).
- Many problems we have seen are optimization problem.
- But we can always recast an optimization problem into a Yes/No problem.

Convert optimization to Yes/No

Optimization Problem

Find the shortest path on a graph G from s to t.

Yes/No version

Is there a path in G from s to t of cost at most k?

Conversion

- Solve the optimization problem.
- Say the shortest path has cost l. Return $l \leq k$.

Conclusion

If you can solve optimization problems fast, you can solve the Yes/No problem fast.

Interestingly

The reverse also holds.

Convert Yes/No to optimization

Yes/No version

Is there a path in G from s to t of cost at most k?

Optimization Problem

Find the shortest path on a graph G from s to t.

Conversion

- Let l be zero and h be the sum of all edge costs
- While $l \leq h$
  - Let $m = \lceil (l + h)/2 \rceil$
  - Is there a path of cost at most m?
    - If yes let $h = m$ and save this path
    - If no let $l = m + 1$
- Return the saved path

Can you do this for all optimization problem?

Yes, as long as you have a lower and upper bound on optimal value.

What is a problem?

Abstract problem Q

A mapping between the set I of problem instances and the set S of problem solutions.

Example: Shortest paths (SP)

An instance of SP is a triple of a graph and two vertices, s and t from this graph. A solution is a sequence of adjacent vertices starting at s and ending at t (Or possible the empty sequence to indicate that no path exists.)

Concrete problem statement

Encoding

An encoding of a set S of objects is a mapping from s to the set $\{0,1\}^*$ (the set of all finite binary strings.)

Example Integers, ASCII

- $0, 1, 2, 3 \rightarrow 000, 001, 010, 011$
- $A, B, C \rightarrow 1000001, 1000010, 1000011$

A concrete problem

Is an encoding of an instance of a problem.

Runtime

Solvable

An algorithm solves a concrete problem in time $O(T(n))$ if, when it is provided by an instance of length n bits, the algorithm can produce the solution in time $O(T(n))$.

Polynomial-time solvable

A concrete problem of length n is polynomial-time solvable if there exists an algorithm to solve it in time $O(n^k)$ for some k.

Warning

We have never used this definition before when talking about runtime. The dependence on the number of bits is crucial. Before we simply used a loosely stated dependence on the input.

Our first definition.

The class $\mathcal{P}$

The set of concrete decision problems that are polynomial-time solvable.

Dependence on encoding

- Consider a problem where the input is a simple integer k.
- And an algorithm running in time $O(k)$.
- If the encoding is unary (an integer k is encoded by k bits) then the runtime is linear
- If the encoding is two-complement, then the number of bits is $\approx \log k$ and the runtime is exponential!

Assumption

From here onward, we will assume a "reasonable" encoding.

Example

Before

Sorting an array of n elements by Heapsort is done in time proportional to $n \log n$.

Now

Sorting an array of n elements each of 64 bits can be done in time bounded above by a polynomial in n.

Focus on polynomial time

Polynomial-time computable

A function $f : \{0,1\}^* \to \{0,1\}^*$ is polynomial-time computable if there exists a polynomial-time algorithm that, given any input $x \in \{0,1\}^*$, produces $f(x)$.

Examples of problems in $\mathcal{P}$

Pretty much all problems we considered during the past months
For example:

- Sorting n integers on 64 bits $(n \log n)$.
- Searching a sorted n long array of 64 bits integers $(\log n)$.
- Matrix multiplication of $n \times n$ matrices of 64 bits floats $(n^{2.7})$.
    - Assuming scalar multiplication of floats is constant.

Examples of problems not known to be in $\mathcal{P}$

- Hard problems:

Examples of problems not known to be in $\mathcal{P}$

- Hard problems:
  - TSP

Examples of problems not known to be in $\mathcal{P}$

- Hard problems:
  - TSP
  - Knapsack

Examples of problems not known to be in $\mathcal{P}$

- Hard problems:
  - TSP
  - Knapsack
  - Subset-sum

Examples of problems not known to be in $\mathcal{P}$

- Hard problems:
  - TSP
  - Knapsack
  - Subset-sum
- Why? We have solved all of the above!

Examples of problems not known to be in $\mathcal{P}$

- Hard problems:
  - TSP
  - Knapsack
  - Subset-sum
- Why? We have solved all of the above!
  - Yes, in time proportional to 2^n (or worse)

Our second definition.

The class $\mathcal{NP}$

A problem P, formulated such that it can be answered with either Yes or No, is in the class $\mathcal{NP}$ if there is a method, given a problem instance and some additional data (called the certificate) to verify in polytime that the answer Yes is correct.

Note

NP does not stand for non-polynomial; it stands for non-deterministic polynomial.

Examples of problems in $\mathcal{NP}$

- Given a network with distances on each arcs.

Examples of problems in $\mathcal{NP}$

- Given a network with distances on each arcs.
  - Problem: Is there a tour of all nodes of cost less than or equal to 42?

Examples of problems in $\mathcal{NP}$

- Given a network with distances on each arcs.
  - Problem: Is there a tour of all nodes of cost less than or equal to 42?
  - Certificate: A an ordered list of all the nodes in the graph.

Examples of problems in $\mathcal{NP}$

- Given a network with distances on each arcs.
  - Problem: Is there a tour of all nodes of cost less than or equal to 42?
  - Certificate: A an ordered list of all the nodes in the graph.
    - Can you write the code to verify in polytime?

Examples of problems in $\mathcal{NP}$

- Given a network with distances on each arcs.
  - Problem: Is there a tour of all nodes of cost less than or equal to 42?
  - Certificate: A an ordered list of all the nodes in the graph.
    - Can you write the code to verify in polytime?
- Given a logical expression with and, or, and negation:

Examples of problems in $\mathcal{NP}$

- Given a network with distances on each arcs.
  - Problem: Is there a tour of all nodes of cost less than or equal to 42?
  - Certificate: A an ordered list of all the nodes in the graph.
    - Can you write the code to verify in polytime?
- Given a logical expression with and, or, and negation:
  - Problem: Is there an assignment of the variables that make the expression True?

Examples of problems in $\mathcal{NP}$

- Given a network with distances on each arcs.
  - Problem: Is there a tour of all nodes of cost less than or equal to 42?
  - Certificate: A an ordered list of all the nodes in the graph.
    - Can you write the code to verify in polytime?
- Given a logical expression with and, or, and negation:
  - Problem: Is there an assignment of the variables that make the expression True?
  - Certificate: An assignment of True/False to the variables that make the expression True.

Examples of problems in $\mathcal{NP}$

- Given a network with distances on each arcs.
  - Problem: Is there a tour of all nodes of cost less than or equal to 42?
  - Certificate: A an ordered list of all the nodes in the graph.
    - Can you write the code to verify in polytime?
- Given a logical expression with and, or, and negation:
  - Problem: Is there an assignment of the variables that make the expression True?
  - Certificate: An assignment of True/False to the variables that make the expression True.
    - i.e.: $A \vee B \vee \neg C$

Examples of problems in $\mathcal{NP}$

- Given a network with distances on each arcs.
  - Problem: Is there a tour of all nodes of cost less than or equal to 42?
  - Certificate: A an ordered list of all the nodes in the graph.
    - Can you write the code to verify in polytime?
- Given a logical expression with and, or, and negation:
  - Problem: Is there an assignment of the variables that make the expression True?
  - Certificate: An assignment of True/False to the variables that make the expression True.
    - i.e.: $A \vee B \vee \neg C$
    - Can you write the code to verify in polytime?

Can you prove that these are in $\mathcal{P}$, in $\mathcal{NP}$?

- Given a graph and weights on the edge, is there a spanning tree of weight at most K?
- Given a graph and weights of the node, is there an independent set of vertices of weight at least k?
- Given a graph, is there a set of edges of size at most k that, if deleted, disconnects the graph?

Stop and think

- We have defined two sets of problems.
  - $\mathcal{P}$
  - $\mathcal{NP}$
- What is the relation, if any, between the two sets?

First set relation

$\mathcal{P} \subseteq \mathcal{NP}$

Since any problem in $\mathcal{P}$ can be solved in polytime, we can ignore whatever certificate is provided.

Open questions

- $\mathcal{P} \subsetneq \mathcal{NP}$? (How would you show this?)

Open questions

- $\mathcal{P} \subsetneq \mathcal{NP}$? (How would you show this?)
- $\mathcal{P} = \mathcal{NP}$?

Our third definition

The class $co-\mathcal{NP}$

A problem P, formulated such that it can be answered with either Yes or No, is in the class $co-\mathcal{NP}$ if there is a method, given a problem instance and some additional data (called the certificate) to verify in polytime that the answer No is correct.

Examples of problems in $co - \mathcal{NP}$

- Given a logical expression with and, or, and negation.

Examples of problems in $co - \mathcal{NP}$

- Given a logical expression with and, or, and negation.
  - Problem: Is the expression a tautology?

Examples of problems in $co-\mathcal{NP}$

- Given a logical expression with and, or, and negation.
  - Problem: Is the expression a tautology?
  - Certificate: An assignment of True/False to the variables that make the expression False.

Examples of problems in $co - \mathcal{NP}$

- Given a logical expression with and, or, and negation.
  - Problem: Is the expression a tautology?
  - Certificate: An assignment of True/False to the variables that make the expression False.
    - Can you write the code to verify in polytime?

Contrast and understand

The class $\mathcal{NP}$

A problem P, formulated such that it can be answered with either Yes or No, is in the class $\mathcal{NP}$ if there is a method, given a problem instance and some additional data (called the certificate) to verify in polytime that the answer Yes is correct.

The class $co - \mathcal{NP}$

A problem P, formulated such that it can be answered with either Yes or No, is in the class $co - \mathcal{NP}$ if there is a method, given a problem instance and some additional data (called the certificate) to verify in polytime that the answer No is correct.

Are there any problems in $co - \mathcal{NP} \cap \mathcal{NP}$?

Discuss

Problems in $co - \mathcal{NP} \cap \mathcal{NP}$?

- All of $\mathcal{P}$

Problems in $co-\mathcal{NP} \cap \mathcal{NP}$?

- All of $\mathcal{P}$
- More interesting: Integer factorization.

Second set relation

$\mathcal{P} \subseteq co - \mathcal{NP}$

Since any problem in $\mathcal{P}$ can be solved in polytime, the certificate can be the empty set.

Open questions

- $\mathcal{P} \subsetneq co - \mathcal{NP}$? (How would you prove this?)

Open questions

- $\mathcal{P} \subsetneq co - \mathcal{NP}$? (How would you prove this?)
- $\mathcal{P} = co - \mathcal{NP} \cap \mathcal{NP}$?

Open questions

- $\mathcal{P} \subsetneq co - \mathcal{NP}$? (How would you prove this?)
- $\mathcal{P} = co - \mathcal{NP} \cap \mathcal{NP}$?
- $\mathcal{NP} = co - \mathcal{NP}$?

Fourth definition

Reducibility

A problem instance Q can be reduced to problem instance Q' if there is a mechanism to construct Q' from Q in such a manner that, if we answer Q', we can extract the answer to Q.
We can also add constraints on the difficulty of the reduction. For instance, requiring that the transformation be done in polytime.

Reducibility

- We use the concept of reducibility to
  - Transform a new problem into a problem we already know how to solve. (A practical application.)
  - Transform a problem A into a problem B to show that A cannot be any more difficult to solve than B. (A theoretical application.)

Practical reducibility

- Exponentiation is trivially reduced to multiplication.
- We considered the problem of computing integer square roots using only the four basic arithmetic operators. This reduces square roots to the basic operations.

Theoretical reducibility

Problem: Hamiltonian cycle (HC)

Given a graph G, is there a Hamiltonian tour, a tour visiting each node exactly once?

Problem: Travelling Salesman tour (TSP)

Given a graph complete K and a set of weights on the edges, what is the smallest cost tour?

Reducible?

Can you transform an instance of HC, in polytime, into an instance of TSP such that, if you solve the TSP, you can answer the HC question in the original graph?

Reducing Hamiltonian cycle to TSP.

Assume we know how to solve TSP and want to solve HC.

- Q: HC on graph G
- Q': Construct the complete graph K with same nodes as G. If edge (i, j) exists in G, put weight 1 on edge (i, j) in G'. Otherwise put weight 2.
- Solve TSP on K. If the minimal tour is of cost $|V|$, then Yes, there is a Hamiltonian tour in G (namely the same one as the TSP tour in K). If the cost is higher, then there is no such tour in G.
- Conclusion: HC is no harder than TSP.

Reducibility

Why do we care? Here is the first reason:

Theorem: If P_1 is polytime reducible to P_2 and $P_2 \in \mathcal{P}$ then $P_1 \in \mathcal{P}$

The main requirement, of course, is that the transformation is done in polytime. This means that if we can reduce a problem to an 'easy' problem, it is also 'easy'.

Fifth definition

The class $\mathcal{NPC}$ (NP-Complete)

A problem P is in class $\mathcal{NPC}$ if it is in $\mathcal{NP}$ and if all other problems in $\mathcal{NP}$ can be reduced to P. (Informally, these are the most difficult problems in $\mathcal{NP}$.)

- This class currently has thousands of problems.

Fifth definition

The class $\mathcal{NPC}$ (NP-Complete)

A problem P is in class $\mathcal{NPC}$ if it is in $\mathcal{NP}$ and if all other problems in $\mathcal{NP}$ can be reduced to P. (Informally, these are the most difficult problems in $\mathcal{NP}$.)

- This class currently has thousands of problems.
- We do not have to do thousands of reductions to add a problem P to this class; we need only to take one problem of this class and reduce it to P.

Fifth definition

The class $\mathcal{NPC}$ (NP-Complete)

A problem P is in class $\mathcal{NPC}$ if it is in $\mathcal{NP}$ and if all other problems in $\mathcal{NP}$ can be reduced to P. (Informally, these are the most difficult problems in $\mathcal{NP}$.)

- This class currently has thousands of problems.
- We do not have to do thousands of reductions to add a problem P to this class; we need only to take one problem of this class and reduce it to P.
- The textbook also defines the class $\mathcal{NP}$ -Hard as the class of problems that satisfy the definition of $\mathcal{NPC}$ except that they may not be in $\mathcal{NP}$.

Reducibility

Why do we care? Here is the second reason:

Theorem: if P_1 is polytime reducible to P_2 and $P_1 \in \mathcal{NPC}$ then $P_2 \in \mathcal{NPC}$.

This, informally, means that P_2 is no easier than P_1. So, if P_1 is 'hard' then P_2 is 'hard'.

The original $\mathcal{NPC}$ problem

- The hardest problem of them all. (i.e. all problems in $\mathcal{NPC}$ can be reduced to it.

The original $\mathcal{NPC}$ problem

- The hardest problem of them all. (i.e. all problems in $\mathcal{NPC}$ can be reduced to it.
- SAT (Satisfiability) (Equivalently, 3-SAT: every clause has 3 terms)

The original $\mathcal{NPC}$ problem

- The hardest problem of them all. (i.e. all problems in $\mathcal{NPC}$ can be reduced to it.
- SAT (Satisfiability) (Equivalently, 3-SAT: every clause has 3 terms)
    - Is there an assignent of the variables that makes the expression True? $A \wedge B \ldots$

The original $\mathcal{NPC}$ problem

- The hardest problem of them all. (i.e. all problems in $\mathcal{NPC}$ can be reduced to it.
- SAT (Satisfiability) (Equivalently, 3-SAT: every clause has 3 terms)
  - Is there an assignent of the variables that makes the expression True? $A \wedge B \ldots$
- To prove this you really need:

The original $\mathcal{NPC}$ problem

- The hardest problem of them all. (i.e. all problems in $\mathcal{NPC}$ can be reduced to it.
- SAT (Satisfiability) (Equivalently, 3-SAT: every clause has 3 terms)
  - Is there an assignent of the variables that makes the expression True? $A \wedge B \ldots$
- To prove this you really need:
  - Language as a tuple (alphabet, states, transition, etc ...)

The original $\mathcal{NPC}$ problem

- The hardest problem of them all. (i.e. all problems in $\mathcal{NPC}$ can be reduced to it.
- SAT (Satisfiability) (Equivalently, 3-SAT: every clause has 3 terms)
  - Is there an assignent of the variables that makes the expression True? $A \wedge B \dots$
- To prove this you really need:
  - Language as a tuple (alphabet, states, transition, etc ...)
  - Turing machine

The original $\mathcal{NPC}$ problem

- The hardest problem of them all. (i.e. all problems in $\mathcal{NPC}$ can be reduced to it.
- SAT (Satisfiability) (Equivalently, 3-SAT: every clause has 3 terms)
  - Is there an assignent of the variables that makes the expression True? $A \wedge B \ldots$
- To prove this you really need:
  - Language as a tuple (alphabet, states, transition, etc ...)
  - Turing machine
- But here is a hand-wavy argument

The original $\mathcal{NPC}$ problem

- The hardest problem of them all. (i.e. all problems in $\mathcal{NPC}$ can be reduced to it.
- SAT (Satisfiability) (Equivalently, 3-SAT: every clause has 3 terms)
  - Is there an assignent of the variables that makes the expression True? $A \wedge B \dots$
- To prove this you really need:
  - Language as a tuple (alphabet, states, transition, etc ...)
  - Turing machine
- But here is a hand-wavy argument
  - We can express Yes/No problems as a set of conditions.

The original $\mathcal{NPC}$ problem

- The hardest problem of them all. (i.e. all problems in $\mathcal{NPC}$ can be reduced to it.
- SAT (Satisfiability) (Equivalently, 3-SAT: every clause has 3 terms)
  - Is there an assignent of the variables that makes the expression True? $A \wedge B \dots$
- To prove this you really need:
  - Language as a tuple (alphabet, states, transition, etc ...)
  - Turing machine
- But here is a hand-wavy argument
  - We can express Yes/No problems as a set of conditions.
  - We can translate these conditions into algebraic expressions.

The original $\mathcal{NPC}$ problem

- The hardest problem of them all. (i.e. all problems in $\mathcal{NPC}$ can be reduced to it.
- SAT (Satisfiability) (Equivalently, 3-SAT: every clause has 3 terms)
  - Is there an assignent of the variables that makes the expression True? $A \wedge B \ldots$
- To prove this you really need:
  - Language as a tuple (alphabet, states, transition, etc . . .)
  - Turing machine
- But here is a hand-wavy argument
  - We can express Yes/No problems as a set of conditions.
  - We can translate these conditions into algebraic expressions.
  - We can translate these algebraic expressions into boolean logic expressions.

The original $\mathcal{NPC}$ problem

- The hardest problem of them all. (i.e. all problems in $\mathcal{NPC}$ can be reduced to it.
- SAT (Satisfiability) (Equivalently, 3-SAT: every clause has 3 terms)
  - Is there an assignent of the variables that makes the expression True? $A \wedge B \ldots$
- To prove this you really need:
  - Language as a tuple (alphabet, states, transition, etc ...)
  - Turing machine
- But here is a hand-wavy argument
  - We can express Yes/No problems as a set of conditions.
  - We can translate these conditions into algebraic expressions.
  - We can translate these algebraic expressions into boolean logic expressions.
  - Therefore all Yes/No problems can be expressed as satisfiability problems.

The original $\mathcal{NPC}$ problem

- The hardest problem of them all. (i.e. all problems in $\mathcal{NPC}$ can be reduced to it.
- SAT (Satisfiability) (Equivalently, 3-SAT: every clause has 3 terms)
  - Is there an assignent of the variables that makes the expression True? $A \wedge B \ldots$
- To prove this you really need:
  - Language as a tuple (alphabet, states, transition, etc ...)
  - Turing machine
- But here is a hand-wavy argument
  - We can express Yes/No problems as a set of conditions.
  - We can translate these conditions into algebraic expressions.
  - We can translate these algebraic expressions into boolean logic expressions.
  - Therefore all Yes/No problems can be expressed as satisfiability problems.
  - Therefore if you can solve satisfiability, you can solve them all!

The original $\mathcal{NPC}$ problem

- The hardest problem of them all. (i.e. all problems in $\mathcal{NPC}$ can be reduced to it.
- SAT (Satisfiability) (Equivalently, 3-SAT: every clause has 3 terms)
  - Is there an assignent of the variables that makes the expression True? $A \wedge B \dots$
- To prove this you really need:
  - Language as a tuple (alphabet, states, transition, etc ...)
  - Turing machine
- But here is a hand-wavy argument
  - We can express Yes/No problems as a set of conditions.
  - We can translate these conditions into algebraic expressions.
  - We can translate these algebraic expressions into boolean logic expressions.
  - Therefore all Yes/No problems can be expressed as satisfiability problems.
  - Therefore if you can solve satisfiability, you can solve them all!
  - (The proof is somewhat more complicated :-))

After SAT ...

- Thousands of problems were shown to be no easier than SAT.
- To show a problem to be in $\mathcal{NPC}$ you need
  - To show it is in $\mathcal{NP}$ (Typically easy)
  - Pick a problem already in $\mathcal{NPC}$ (Thousands to choose from)
  - Reduce it to your "new" problem. (Needs clever constructions)
  - Show the reduction can be done in polytime. (Usually easy)
  - Show that an answer to one problem provides an answer to the other.

Clique is in $\mathcal{NPC}$

- Clique problem: Given an undirected graph $G = (V, E)$ and an integer k, is there a subgraph of G forming a complete graph on k vertices?

Clique is in $\mathcal{NPC}$

- Clique problem: Given an undirected graph $G = (V, E)$ and an integer k, is there a subgraph of G forming a complete graph on k vertices?
- Show it is in $\mathcal{NP}$

Clique is in $\mathcal{NPC}$

- Clique problem: Given an undirected graph $G = (V, E)$ and an integer k, is there a subgraph of G forming a complete graph on k vertices?
- Show it is in $\mathcal{NP}$
  - Given a set of $V' \subset V$ vertices of size k in G, we can check, given an appropriate data structure, that all vertices in v' are connected to each other.

Clique is in $\mathcal{NPC}$

- Clique problem: Given an undirected graph $G = (V, E)$ and an integer k, is there a subgraph of G forming a complete graph on k vertices?
- Show it is in $\mathcal{NP}$
  - Given a set of $V' \subset V$ vertices of size k in G, we can check, given an appropriate data structure, that all vertices in v' are connected to each other.
- Reducibility from SAT (actually from 3-SAT, which is equivalent)

Clique is in $\mathcal{NPC}$

- Clique problem: Given an undirected graph $G = (V, E)$ and an integer k, is there a subgraph of G forming a complete graph on k vertices?
- Show it is in $\mathcal{NP}$
  - Given a set of $V' \subset V$ vertices of size k in G, we can check, given an appropriate data structure, that all vertices in v' are connected to each other.
- Reducibility from SAT (actually from 3-SAT, which is equivalent)
  - Consider a boolean formula $\phi = C_1 \wedge C_2 \wedge \ldots C_k$

Clique is in $\mathcal{NPC}$

- Clique problem: Given an undirected graph $G = (V, E)$ and an integer k, is there a subgraph of G forming a complete graph on k vertices?
- Show it is in $\mathcal{NP}$
  - Given a set of $V' \subset V$ vertices of size k in G, we can check, given an appropriate data structure, that all vertices in v' are connected to each other.
- Reducibility from SAT (actually from 3-SAT, which is equivalent)
  - Consider a boolean formula $\phi = C_1 \wedge C_2 \wedge \ldots C_k$
  - With $C_r = l_1^r \vee l_2^r \vee l_3^r$

Clique is in $\mathcal{NPC}$

- Clique problem: Given an undirected graph $G = (V, E)$ and an integer k, is there a subgraph of G forming a complete graph on k vertices?
- Show it is in $\mathcal{NP}$
  - Given a set of $V' \subset V$ vertices of size k in G, we can check, given an appropriate data structure, that all vertices in v' are connected to each other.
- Reducibility from SAT (actually from 3-SAT, which is equivalent)
  - Consider a boolean formula $\phi = C_1 \wedge C_2 \wedge \ldots C_k$
  - With $C_r = l_1^r \vee l_2^r \vee l_3^r$
  - Construct a graph $G = (V, E)$

Clique is in $\mathcal{NPC}$

- Clique problem: Given an undirected graph $G = (V, E)$ and an integer k, is there a subgraph of G forming a complete graph on k vertices?
- Show it is in $\mathcal{NP}$
  - Given a set of $V' \subset V$ vertices of size k in G, we can check, given an appropriate data structure, that all vertices in v' are connected to each other.
- Reducibility from SAT (actually from 3-SAT, which is equivalent)
  - Consider a boolean formula $\phi = C_1 \wedge C_2 \wedge \ldots C_k$
  - With $C_r = l_1^r \vee l_2^r \vee l_3^r$
  - Construct a graph $G = (V, E)$
  - For every triple l_1^r, l_2^r, l_3^r add three nodes v_1^r, v_2^r, v_3^r

Clique is in $\mathcal{NPC}$

- Clique problem: Given an undirected graph $G = (V, E)$ and an integer k, is there a subgraph of G forming a complete graph on k vertices?
- Show it is in $\mathcal{NP}$
  - Given a set of $V' \subset V$ vertices of size k in G, we can check, given an appropriate data structure, that all vertices in v' are connected to each other.
- Reducibility from SAT (actually from 3-SAT, which is equivalent)
  - Consider a boolean formula $\phi = C_1 \wedge C_2 \wedge \ldots C_k$
  - With $C_r = l_1^r \vee l_2^r \vee l_3^r$
  - Construct a graph $G = (V, E)$
  - For every triple l_1^r, l_2^r, l_3^r add three nodes v_1^r, v_2^r, v_3^r
  - Add edge (v_i^r, v_j^s) if

Clique is in $\mathcal{NPC}$

- Clique problem: Given an undirected graph $G = (V, E)$ and an integer k, is there a subgraph of G forming a complete graph on k vertices?
- Show it is in $\mathcal{NP}$
  - Given a set of $V' \subset V$ vertices of size k in G, we can check, given an appropriate data structure, that all vertices in v' are connected to each other.
- Reducibility from SAT (actually from 3-SAT, which is equivalent)
  - Consider a boolean formula $\phi = C_1 \wedge C_2 \wedge \ldots C_k$
  - With $C_r = l_1^r \vee l_2^r \vee l_3^r$
  - Construct a graph $G = (V, E)$
  - For every triple l_1^r, l_2^r, l_3^r add three nodes v_1^r, v_2^r, v_3^r
  - Add edge (v_i^r, v_j^s) if
    - $r \neq s$

Clique is in $\mathcal{NPC}$

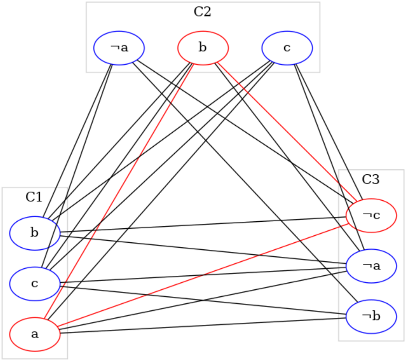
- Clique problem: Given an undirected graph $G = (V, E)$ and an integer k, is there a subgraph of G forming a complete graph on k vertices?
- Show it is in $\mathcal{NP}$
  - Given a set of $V' \subset V$ vertices of size k in G, we can check, given an appropriate data structure, that all vertices in v' are connected to each other.
- Reducibility from SAT (actually from 3-SAT, which is equivalent)
  - Consider a boolean formula $\phi = C_1 \wedge C_2 \wedge \ldots C_k$
  - With $C_r = l_1^r \vee l_2^r \vee l_3^r$
  - Construct a graph $G = (V, E)$
  - For every triple l_1^r, l_2^r, l_3^r add three nodes v_1^r, v_2^r, v_3^r
  - Add edge (v_i^r, v_j^s) if
    - $r \neq s$
    - $l_i^r \neq \neg l_j^s$ (they are consistent)

Clique is in $\mathcal{NPC}$

- Clique problem: Given an undirected graph $G = (V, E)$ and an integer k, is there a subgraph of G forming a complete graph on k vertices?
- Show it is in $\mathcal{NP}$
  - Given a set of $V' \subset V$ vertices of size k in G, we can check, given an appropriate data structure, that all vertices in v' are connected to each other.
- Reducibility from SAT (actually from 3-SAT, which is equivalent)
  - Consider a boolean formula $\phi = C_1 \wedge C_2 \wedge \dots C_k$
  - With $C_r = l_1^r \vee l_2^r \vee l_3^r$
  - Construct a graph $G = (V, E)$
  - For every triple l_1^r, l_2^r, l_3^r add three nodes v_1^r, v_2^r, v_3^r
  - Add edge (v_i^r, v_j^s) if
    - $r \neq s$
    - $l_i^r \neq \neg l_j^s$ (they are consistent)
  - Claim: ϕ is satisfiable if and only if G has a clique of size k.

Clique is in $\mathcal{NPC}$

- Consider $(a \vee b \vee c) \wedge (\neg a \vee b \vee c) \wedge (\neg a \vee \neg b \neg c)$



- The red clique indicates $a, b, \neg c$ true, satisfying the expression.

Vertex cover is in $\mathcal{NPC}$

- Given a graph, is there a set of nodes S of size at least k such that every edge of the graph has at least one vertex in S?

Vertex cover is in $\mathcal{NPC}$

- Given a graph, is there a set of nodes S of size at least k such that every edge of the graph has at least one vertex in S?
- Show it is in $\mathcal{NP}$

Vertex cover is in $\mathcal{NPC}$

- Given a graph, is there a set of nodes S of size at least k such that every edge of the graph has at least one vertex in S?
- Show it is in $\mathcal{NP}$
  - Given a set of S vertices of size k, we can check, by going over every edge, that each edge has an end in S

Vertex cover is in $\mathcal{NPC}$

- Given a graph, is there a set of nodes S of size at least k such that every edge of the graph has at least one vertex in S?
- Show it is in $\mathcal{NP}$
  - Given a set of S vertices of size k, we can check, by going over every edge, that each edge has an end in S
- Reducibility from Clique (Does G have a clique of size k?)

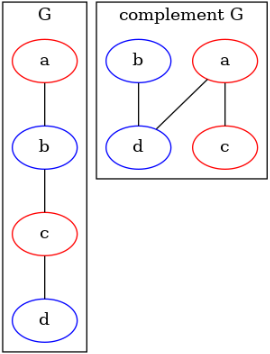
Vertex cover is in $\mathcal{NPC}$

- Given a graph, is there a set of nodes S of size at least k such that every edge of the graph has at least one vertex in S?
- Show it is in $\mathcal{NP}$
  - Given a set of S vertices of size k, we can check, by going over every edge, that each edge has an end in S
- Reducibility from Clique (Does G have a clique of size k?)
  - Construct the complement of G (put edges where there are none and delete current edge)

Vertex cover is in $\mathcal{NPC}$

- Given a graph, is there a set of nodes S of size at least k such that every edge of the graph has at least one vertex in S?
- Show it is in $\mathcal{NP}$
  - Given a set of S vertices of size k, we can check, by going over every edge, that each edge has an end in S
- Reducibility from Clique (Does G have a clique of size k?)
  - Construct the complement of G (put edges where there are none and delete current edge)
  - Claim: G has a clique of size k if and only if $\overline{G}$ has a vertex cover of size $|V| - k$.

Vertex cover is in $\mathcal{NPC}$



- Vertex cover of size 2 in G : Clique of size $4 - 2$ in $\overline{G}$
- No Vertex cover of size 1 in G : No Clique of size $4 - 1$ in $\overline{G}$

Subset sum is in $\mathcal{NPC}$

- Given a set S of integers and a target T, is there a subset of S summing to T?

Subset sum is in $\mathcal{NPC}$

- Given a set S of integers and a target T, is there a subset of S summing to T?
- Show it is in $\mathcal{NP}$

Subset sum is in $\mathcal{NPC}$

- Given a set S of integers and a target T, is there a subset of S summing to T?
- Show it is in $\mathcal{NP}$
  - Given the subset, just sum everything and verify that the sum is T.

Subset sum is in $\mathcal{NPC}$

- Given a set S of integers and a target T, is there a subset of S summing to T?
- Show it is in $\mathcal{NP}$
  - Given the subset, just sum everything and verify that the sum is T.
- Reducibility from Vertex Cover

Subset sum is in $\mathcal{NPC}$

- Given a set S of integers and a target T, is there a subset of S summing to T?
- Show it is in $\mathcal{NP}$
  - Given the subset, just sum everything and verify that the sum is T.
- Reducibility from Vertex Cover
  - Given a graph $G = (V, E)$ and and integer k, is there a vertex cover of size k?

Subset sum is in $\mathcal{NPC}$

- Given a set S of integers and a target T, is there a subset of S summing to T?
- Show it is in $\mathcal{NP}$
  - Given the subset, just sum everything and verify that the sum is T.
- Reducibility from Vertex Cover
  - Given a graph $G = (V, E)$ and and integer k, is there a vertex cover of size k?
  - Consider the node-arc incidence matrix of the graph G.

Subset sum is in $\mathcal{NPC}$

- Given a set S of integers and a target T, is there a subset of S summing to T?
- Show it is in $\mathcal{NP}$
  - Given the subset, just sum everything and verify that the sum is T.
- Reducibility from Vertex Cover
  - Given a graph $G = (V, E)$ and and integer k, is there a vertex cover of size k?
  - Consider the node-arc incidence matrix of the graph G.
  - Embed this matrix into a larger matrix by

Subset sum is in $\mathcal{NPC}$

- Given a set S of integers and a target T, is there a subset of S summing to T?
- Show it is in $\mathcal{NP}$
  - Given the subset, just sum everything and verify that the sum is T.
- Reducibility from Vertex Cover
  - Given a graph $G = (V, E)$ and and integer k, is there a vertex cover of size k?
  - Consider the node-arc incidence matrix of the graph G.
  - Embed this matrix into a larger matrix by
    - Adding an identity matrix below it.

Subset sum is in $\mathcal{NPC}$

- Given a set S of integers and a target T, is there a subset of S summing to T?
- Show it is in $\mathcal{NP}$
  - Given the subset, just sum everything and verify that the sum is T.
- Reducibility from Vertex Cover
  - Given a graph $G = (V, E)$ and and integer k, is there a vertex cover of size k?
  - Consider the node-arc incidence matrix of the graph G.
  - Embed this matrix into a larger matrix by
    - Adding an identity matrix below it.
    - Prepend a column with 1 for every row of the incidence matrix and 0 for the identity row.

Subset sum is in $\mathcal{NPC}$

- Given a set S of integers and a target T, is there a subset of S summing to T?
- Show it is in $\mathcal{NP}$
  - Given the subset, just sum everything and verify that the sum is T.
- Reducibility from Vertex Cover
  - Given a graph $G = (V, E)$ and and integer k, is there a vertex cover of size k?
  - Consider the node-arc incidence matrix of the graph G.
  - Embed this matrix into a larger matrix by
    - Adding an identity matrix below it.
    - Prepend a column with 1 for every row of the incidence matrix and 0 for the identity row.
    - Read every row $0 \leq i \leq |V| - 1$ as $4^{|E|} + \sum_{j=0}^{|E|-1} b_{ij} 4^j$

Subset sum is in $\mathcal{NPC}$

- Given a set S of integers and a target T, is there a subset of S summing to T?
- Show it is in $\mathcal{NP}$
  - Given the subset, just sum everything and verify that the sum is T.
- Reducibility from Vertex Cover
  - Given a graph $G = (V, E)$ and and integer k, is there a vertex cover of size k?
  - Consider the node-arc incidence matrix of the graph G.
  - Embed this matrix into a larger matrix by
    - Adding an identity matrix below it.
    - Prepend a column with 1 for every row of the incidence matrix and 0 for the identity row.
    - Read every row $0 \leq i \leq |V| - 1$ as $4^{|E|} + \sum_{j=0}^{|E|-1} b_{ij}4^j$
    - Read every row $|V| \leq i \leq 2|V| - 1$ as $\sum_{j=0}^{|E|-1} b_{ij}4^j$

Subset sum is in $\mathcal{NPC}$

- Given a set S of integers and a target T, is there a subset of S summing to T?
- Show it is in $\mathcal{NP}$
  - Given the subset, just sum everything and verify that the sum is T.
- Reducibility from Vertex Cover
  - Given a graph $G = (V, E)$ and and integer k, is there a vertex cover of size k?
  - Consider the node-arc incidence matrix of the graph G.
  - Embed this matrix into a larger matrix by
    - Adding an identity matrix below it.
    - Prepend a column with 1 for every row of the incidence matrix and 0 for the identity row.
    - Read every row $0 \leq i \leq |V| - 1$ as $4^{|E|} + \sum_{j=0}^{|E|-1} b_{ij} 4^j$
    - Read every row $|V| \leq i \leq 2|V| - 1$ as $\sum_{j=0}^{|E|-1} b_{ij} 4^j$
    - Construct $T = k4^{|E|} + \sum_{j=0}^{|E|-1} 2 \cdot 4^j$

Subset sum is in $\mathcal{NPC}$

- Given a set S of integers and a target T, is there a subset of S summing to T?
- Show it is in $\mathcal{NP}$
  - Given the subset, just sum everything and verify that the sum is T.
- Reducibility from Vertex Cover
  - Given a graph $G = (V, E)$ and and integer k, is there a vertex cover of size k?
  - Consider the node-arc incidence matrix of the graph G.
  - Embed this matrix into a larger matrix by
    - Adding an identity matrix below it.
    - Prepend a column with 1 for every row of the incidence matrix and 0 for the identity row.
    - Read every row $0 \leq i \leq |V| - 1$ as $4^{|E|} + \sum_{j=0}^{|E|-1} b_{ij} 4^j$
    - Read every row $|V| \leq i \leq 2|V| - 1$ as $\sum_{j=0}^{|E|-1} b_{ij} 4^j$
    - Construct $T = k4^{|E|} + \sum_{j=0}^{|E|-1} 2 \cdot 4^j$
  - Claim: G has a vertex cover of size k iff there is a subset of the rows summing to T.

Subset sum example

- Consider the graph
$(x_0, x_1, x_2, x_3, x_4, (x_0, x_3), (x_0, x_4), (x_1, x_2), (x_1, x_4), (x_2, x_4))$
- Its incidence matrix is $\begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix}$

Subset sum example

- Graph
  $G = (\{x_0, x_1, x_2, x_3, x_4\}, \{(x_0, x_3), (x_0, x_4), (x_1, x_2), (x_1, x_4), (x_2, x_4)\})$
- The embedding is
$$\begin{bmatrix}
x_0 & = & 1 & 0 & 0 & 1 & 0 & 1 & = & 1041 \\
x_1 & = & 1 & 1 & 0 & 0 & 1 & 0 & = & 1284 \\
x_2 & = & 1 & 1 & 1 & 0 & 0 & 0 & = & 1344 \\
x_3 & = & 1 & 0 & 0 & 1 & 0 & 0 & = & 1040 \\
x_4 & = & 1 & 0 & 1 & 0 & 1 & 1 & = & 1093 \\
 & & 0 & 1 & 0 & 0 & 0 & 0 & = & 256 \\
 & & 0 & 0 & 1 & 0 & 0 & 0 & = & 64 \\
 & & 0 & 0 & 0 & 1 & 0 & 0 & = & 16 \\
 & & 0 & 0 & 0 & 0 & 1 & 0 & = & 4 \\
 & & 0 & 0 & 0 & 0 & 0 & 1 & = & 1 \\
T & = & 3 & 2 & 2 & 2 & 2 & 2 & = & 3754
\end{bmatrix}$$
- G has a VC of size 3 iff there is a subset of $\{1041, 1284, \ldots, 1\}$ summing to 3754.

What we have shown

- SAT is as hard as anything in $\mathcal{NP}$ hence in $\mathcal{NPC}$
- Clique is no easier than SAT
- Vertex Cover is no easier than Clique
- Subset Sum is no easier than Vertex Cover
- Hence all of the above are in $\mathcal{NPC}$

Third set relation

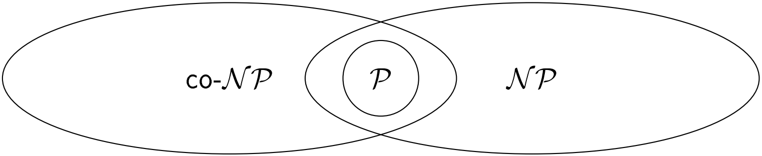
$\mathcal{NPC} \subseteq \mathcal{NP}$

By definition.

Open problems

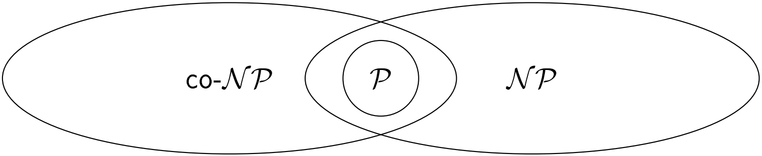
- $\mathcal{P} = \mathcal{NP} = co - \mathcal{NP} = \mathcal{NPC}?$

Fact (though very misleading picture)



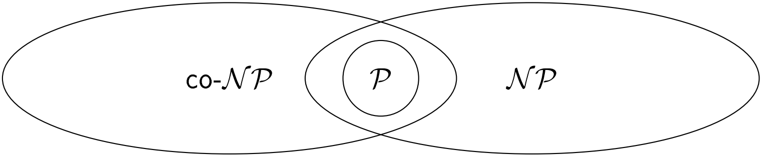
- Why misleading?

Fact (though very misleading picture)



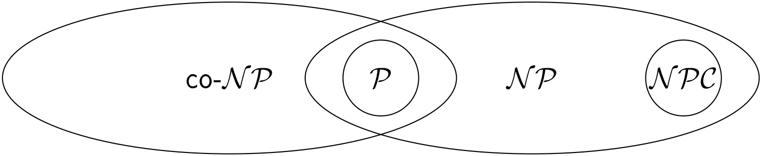
- Why misleading?
  - Because there is not a single instance of a problem known to be in the bubbles but not in $\mathcal{P}$.

Fact (though very misleading picture)

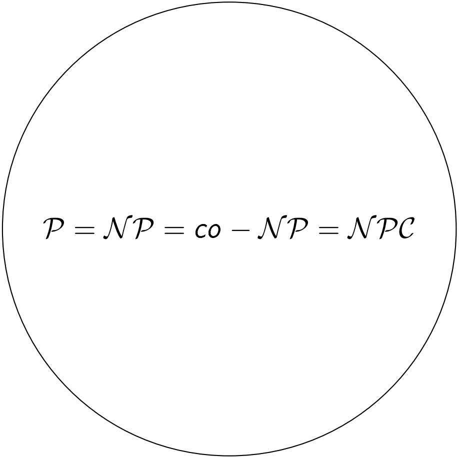


- Why misleading?
  - Because there is not a single instance of a problem known to be in the bubbles but not in $\mathcal{P}$.
- Where is $\mathcal{NPC}$?

Conjecture ($\mathcal{P} \neq \mathcal{NP}$)



Conjecture ($\mathcal{P} = \mathcal{NP}$)



Conjecture

So I'll state, as one of the few definite conclusions of this survey, that $\mathcal{P} = \mathcal{NP}$? is either true or false. It's one or the other. But we may not be able to prove which way it goes, and we may not be able to prove that we can't prove it.
– Lance Fortnow, University of Chicago

This is not the full story

Have a look

https://complexityzoo.uwaterloo.ca/Complexity_Zoo

Final note

- There are problems harder than $\mathcal{NPC}$.
- We ignored space.

Questions

- What would you conclude if I proved that TSP is in $\mathcal{P}$?
- What would you conclude if I proved that TSP is not in $\mathcal{P}$?