So far the algorithms we have considered have all been polynomial in the input size.

- For example, $O(nm^2), O(n), O(n^2), O(n \log n)$.

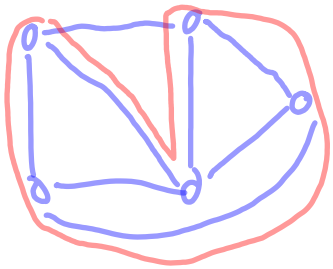
These running times allow for fairly large inputs to be solved efficiently in practice.

Can all (or most) interesting computational problems be solved in polynomial time?

It seems unlikely. Some problems may be too difficult.

An example of a difficult problem is the **traveling salesman problem** (TSP).

- Input: undirected graph with weights on the edges.
- Output: a shortest cycle which visits each vertex exactly once.

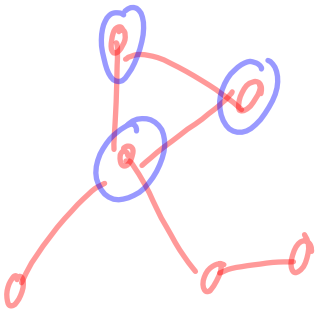


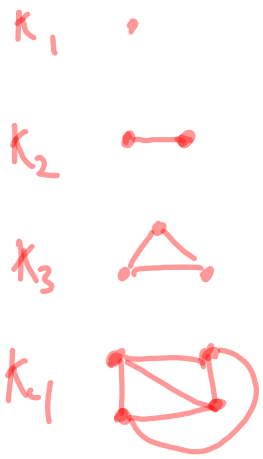
Idea: There are some set of cities that a salesman needs to visit, and the output will produce the most efficient way to visit each of the cities.

The best known algorithm: $O(n2^n)$ (exponential in the size of the input).

Another example of a difficult problem is the **clique** problem.

- Input: undirected graph $G = (V, E)$.
- Output: largest subset C of V such that every pair of vertices in C has an edge between them (C is called a *clique*).





Again, the best known algorithm is $O(n2^n)$.

It would be great if we could design a polynomial time algorithm for these difficult problems. Research has tried for decades and so far has not been able to design such an algorithm.

This begs the question as to whether we could prove that it is not possible to design such an algorithm.

- While it would be a fantastic result which most researchers believe is more likely to be true, proving this seems to be extremely difficult.
- The best known lower bound for the previously mentioned problems is $4.5n$.

So is there anything else we can do with problems which seem extremely difficult?

Research has shown that it is possible to show that some problems are equivalently difficult (we will consider examples of this later in this lecture).

- If a polynomial time algorithm is given for one problem, then all of these "equivalent" problems would also have a polynomial time algorithm.
- If one of them is shown to not have a polynomial time algorithm, then all of the problems would not have a polynomial time algorithm.

If we can show that a problem Π is equivalent to other well studied problems without efficient algorithms, then that strongly suggests that Π does not have an efficient algorithm either.

This technique has worked for over 10,000 hard problems.

A *decision problem* is a problem in which we want to determine the answer to a yes/no question regarding the input.

An example of a decision problem using clique:

- Input: an undirected graph G and a positive integer k.
- Question: Does G contain a clique of size at least k?
- Output: Yes or No.

We could also consider *optimization* variants of the clique problem:

1. What is the largest k such that G contains a clique of size k?
2. What is the largest clique of G?

The decision problem is the easiest to compute, the first optimization variant is the next easiest, and the second optimization variant is the hardest to compute.

But we can show that if we can solve the decision prob in poly. time, then we can solve the optimization problems in poly time.

If we can solve decision prob in poly time, run this algorithm for each $k \in \{1, \ldots n\}$ and output the largest k s.t. the alg returned "Yes".

If we can solve 1st opt. problem, then find the size of the largest clique. Delete an edge, and again find the size of the largest clique. If it stays the same, edge is not in the maximum clique. If it decreases, then it is in the max clique. Repeat to find all edges.

Decision problems belong to an important class of problems known as **NP**.

Problems in NP can all be solved in *non-deterministic polynomial* time.

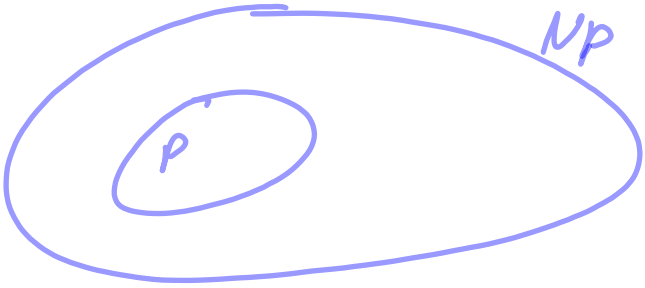
Intuitively, a solution can be *verified* in polynomial time.

- For example, if someone gives us a subset of a graph C, we can verify if it is a clique of size at least k in polynomial time.
- Also, if someone gives us a TSP tour T, we can verify if its length is at most k in polynomial time.

This implies that the decision variants of clique and TSP are in NP.

Another important class of problems is **P**. If a decision problem Π can be solved by a polynomial time algorithm, then Π is in P.

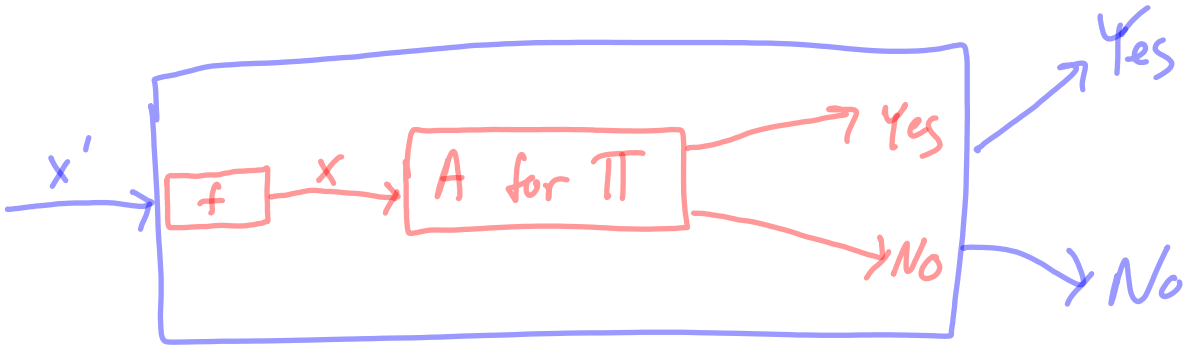
Every problem that is in P is also in NP (i.e. P is a subset of NP). If the problem can be solved in polynomial time, then clearly a solution can be verified to be correct in polynomial time.



A major open question (perhaps the largest open question in all of computer science), is whether there exists a problem that is in NP that is *not* in P. In other words, does P = NP?

Now we will discuss how we can show that a problem is equivalent to another problem.

Reduction: Π' to Π



Π' is *polynomial time reducible* to Π ($\Pi' \leq \Pi$) iff there is a polynomial time function f that maps inputs x' for Π' into inputs x for Π such that for any x' we have $\Pi'(x') = \Pi(f(x'))$.

Intuitively, we need the following to hold true:

- We can convert x' into an input x such that x' is a "yes instance" for Π' iff x is a "yes instance" for Π.

If $\Pi' \leq \Pi$ then the following facts are true:

1. if Π is in P, then so is Π'.
2. if Π is in NP, then so is Π'.
3. if $\Pi'' \leq \Pi'$, then $\Pi'' \leq \Pi$ (transitivity).

An *independent set* of a graph is a subset of the vertices such that no pair of vertices has an edge between them.

The independent set problem (IS) is to determine if a graph G has an independent set of size at least k.

We will show that Clique $\leq$ IS.

- We show that any input $(G = (V, E), k)$ to the clique problem can be converted in polynomial time to an input $(G' = (V', E'), k')$ to the IS problem.
- It will be done in a way so that G has a clique of size at least k iff G' has an independent set of size at least k'.

Construction:

- $k' = k$
- $V' = V$, $E' = \overline{E}$

The key observation is that cliques and independent sets are "complements" of each other w.r.t. edges. Therefore we can obtain the reduction by allowing E' to be the "complement" of E. If C is a clique in G, then the same vertices will be an independent set in G'.

G: G'

We now have a method of showing that one problem Π' at worst as hard as another problem Π.

If we can prove the reduction in both directions (i.e. show $\Pi' \leq \Pi$ and $\Pi \leq \Pi'$), then a polynomial time algorithm for one problem would imply a polynomial time algorithm for the other as well.

Suppose there was a problem Π such that for any Π' in NP, it was true that $\Pi' \leq \Pi$? Then to show that a problem Π' in NP was hard, we would only need to show $\Pi \leq \Pi'$ (as $\Pi' \leq \Pi$ would be implied by Π' being in NP).