Computational Complexity; slides 1, HT 2022
Introduction, motivation

Prof. Paul W. Goldberg (Dept. of Computer Science, University of Oxford)

HT 2022

Administrative notes

It is nice to return to teaching in-person!

www.cs.ox.ac.uk/people/paul.goldberg/CC/index.html

- Slides, exercise sheets, often updated

www.cs.ox.ac.uk/teaching/courses/2021-2022/complexity/

- General info

Problem sheets: classes planned for weeks 3 – 8
Available on web page by Monday of previous week
check hand-in deadlines

Aims

(from the web page)

- Introduce the most important complexity classes
- Give you tools to classify problems into appropriate complexity classes
- Enable you to reduce one problem to another

Above terminology to be made precise

We will see there are major gaps in our understanding of computation!

here, mostly focus on time/space requirements; there is also "communication complexity", "query complexity", ...

note usage of word "complexity"

Background, other courses

Hopefully you know about algorithms, big-O notation, "problem", TM, propositional logic. See e.g. chapter 7.1 in Sipser's textbook
Part A courses:

- Models of Computation:
  - Introduce Turing machines as a universal computing device
  - Classification of problems into decidable/undecidable
  - further classification of undecidable problems
- Logic and Proof:
  - SAT, CNF, etc
- Algorithms (part A)
  - address "intractability" studied here
- Design and Analysis of Algorithms (prelims)
  - design of efficient algorithms.
  - asymptotic complexity analysis of runtime.

Polynomial-time computation, the class **P**

problems solvable in time $O(n)$, $O(n \log n)$, $O(n^{10})$, ...

Given a novel problem, usual Q1: is it in **P**?

Why do we like this concept?

- nice closure/composition properties
  composition of 2 poly-time algorithms is poly-time
- **P** works surprisingly well as a model of "efficiently computable", "fast algorithm"
  If a problem is solvable in time $O(n^{100})$, usually it has a genuinely efficient algorithm
- We can ignore details of model of computation; "clean" analysis
- poly-time algorithms highlight structure of a problem they solve (quote on next slide)

"poly-time" not just about computational efficiency

Poly-time algorithm tells you about the structure of a problem. Contrast with "brute-force" algorithm

A relevant quote (context: looking for "equilibrium prices" in markets)

What do we learn by proving that an equilibrium computation problem is "difficult" in a complexity-theoretic sense? First, assuming widely believed mathematical conjectures, it implies that there will never be a fast, general-purpose computational method for solving the problem. Second, it rules out many structural results, such as convexity or duality, that are often used to understand and justify economic models.

Tim Roughgarden: Computing equilibria: a computational complexity perspective *Economic Theory* (2010)

Some problems don't seem to have efficient algorithms

is given it is quite another matter to determine its factors. Can the reader say what two numbers multiplied together will produce the number 8,616,460,799? I think it unlikely that anyone but myself will ever know; for they are two large prime numbers, and can only be re-discovered by trying in succession a long series of prime divisors until the right one be fallen upon. The work would probably occupy a good computer for many weeks, but it did not occupy me many minutes to multiply the two factors together. Similarly there is no direct process for discovering whether any number is a prime or not; it is only by exhaustively trying all inferior numbers which could be divisors, that we can show there is none, and the labour of the process would be intolerable were it not performed systematically once for all in the process known as the Sieve of Eratosthenes, the results being registered in tables of prime numbers.

Road map (roughly)

1. **[2 lectures]** introduction, Turing machines, (un)decidability, reductions

move swiftly from qualitative to quantitative considerations:

2. **[1 lecture] Deterministic Complexity Classes**. DTIME[t]. Linear Speed-up Theorem. PTime. Polynomial reducibility.
3. **[3 lectures] NP, co-NP, (co-)NP-completeness**. Non-deterministic Turing machines. NTIME[t]. Polynomial time verification. NP-completeness. Cook-Levin Theorem.
4. **[3 lectures] Space complexity and hierarchy theorems**. DSPACE[s]. Linear Space Compression Theorem. PSPACE, NPSPACE. PSPACE = NPSPACE. PSPACE-completeness. Quantified Boolean Formula problem is PSPACE-complete. L, NL and NL-completeness. NL = coNL. Hierarchy theorems.

Road map

5. **[2 lectures] Randomized Complexity**. The classes BPP, RP, ZPP. Interactive proof systems: IP = PSPACE.
6. **Advanced topics.** Randomised complexity, Circuit complexity, total search

Reading List

Primary:

- S. Arora and B. Barak, *Computational Complexity: A Modern Approach*, Cambridge University Press
- M. Sipser, *Introduction to the Theory of Computation*, 2005

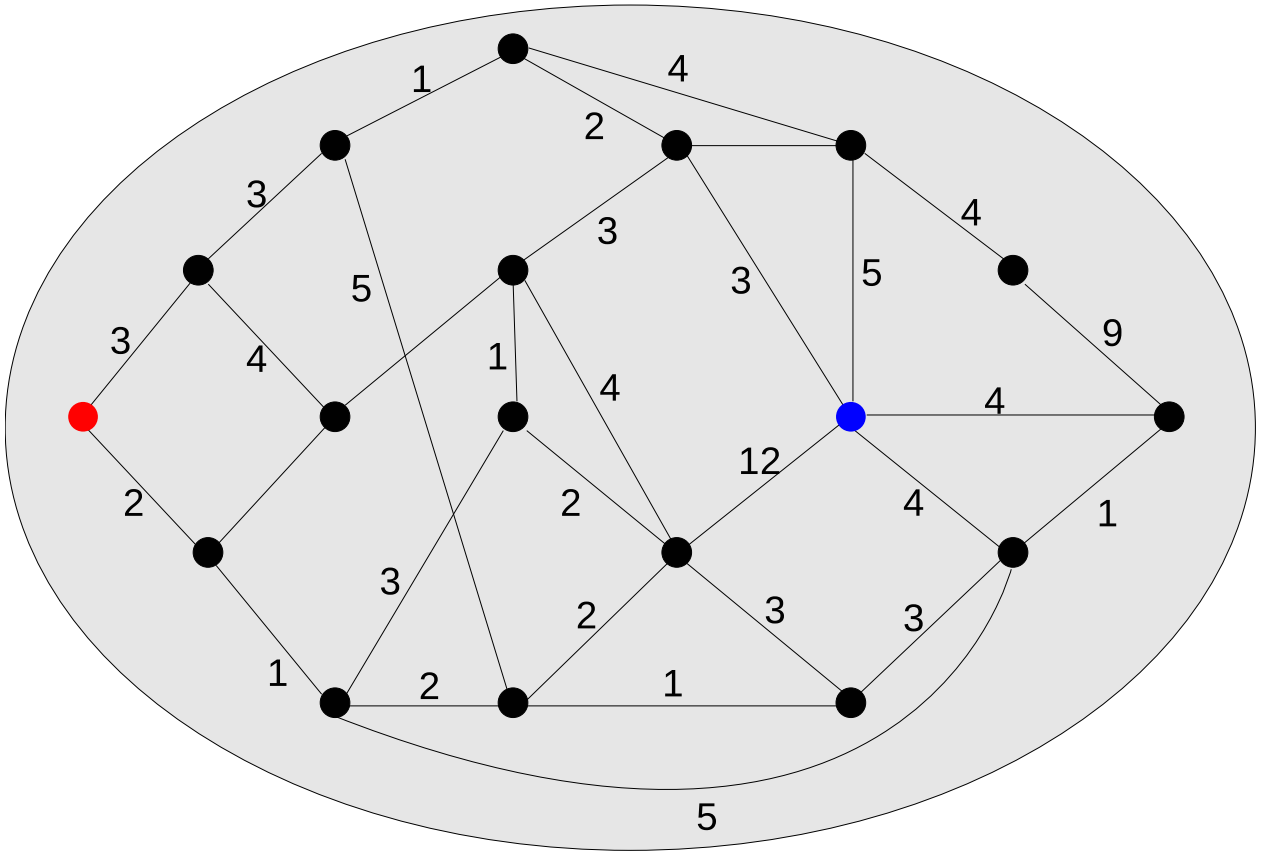
Further:

- C.H. Papadimitriou, *Computational Complexity*, 1994.
- I. Wegener, *Complexity Theory*, Springer, 2005.
- O. Goldreich, *Complexity Theory*, CUP, 2008.
- M.R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 1979.
- T.H. Cormen, S. Clifford, C.E. Leiserson and R. L. Rivest, *Introduction to Algorithms*, 2001.

Example: shortest path problem

SHORTEST PATH

Given a weighted graph and two vertices s, t, find a *shortest path* between s and t.

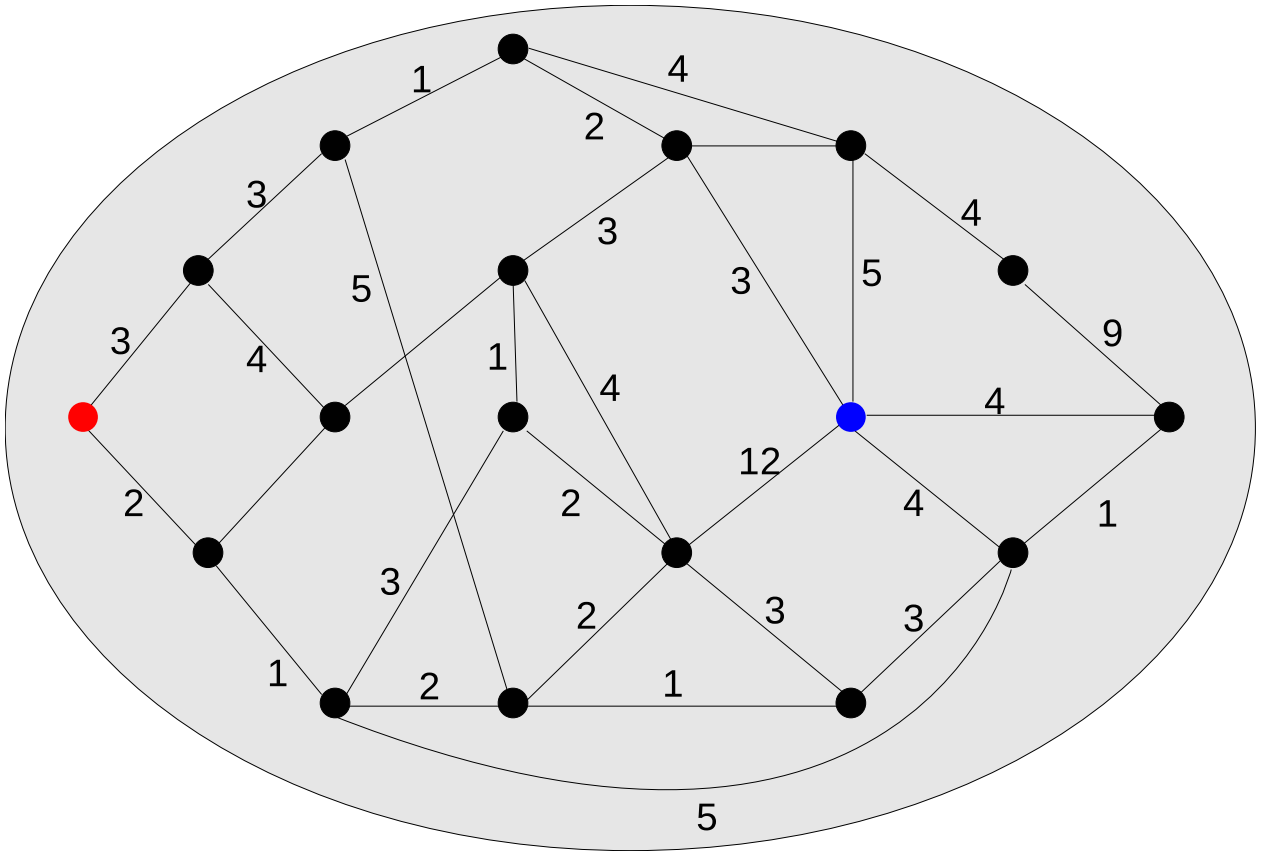


Can be solved efficiently (for instance with Dijkstra's algorithm)

related example: longest path

LONGEST PATH

Given a weighted graph and two vertices s, t, find a *longest simple (cycle-free) path* between s and t.

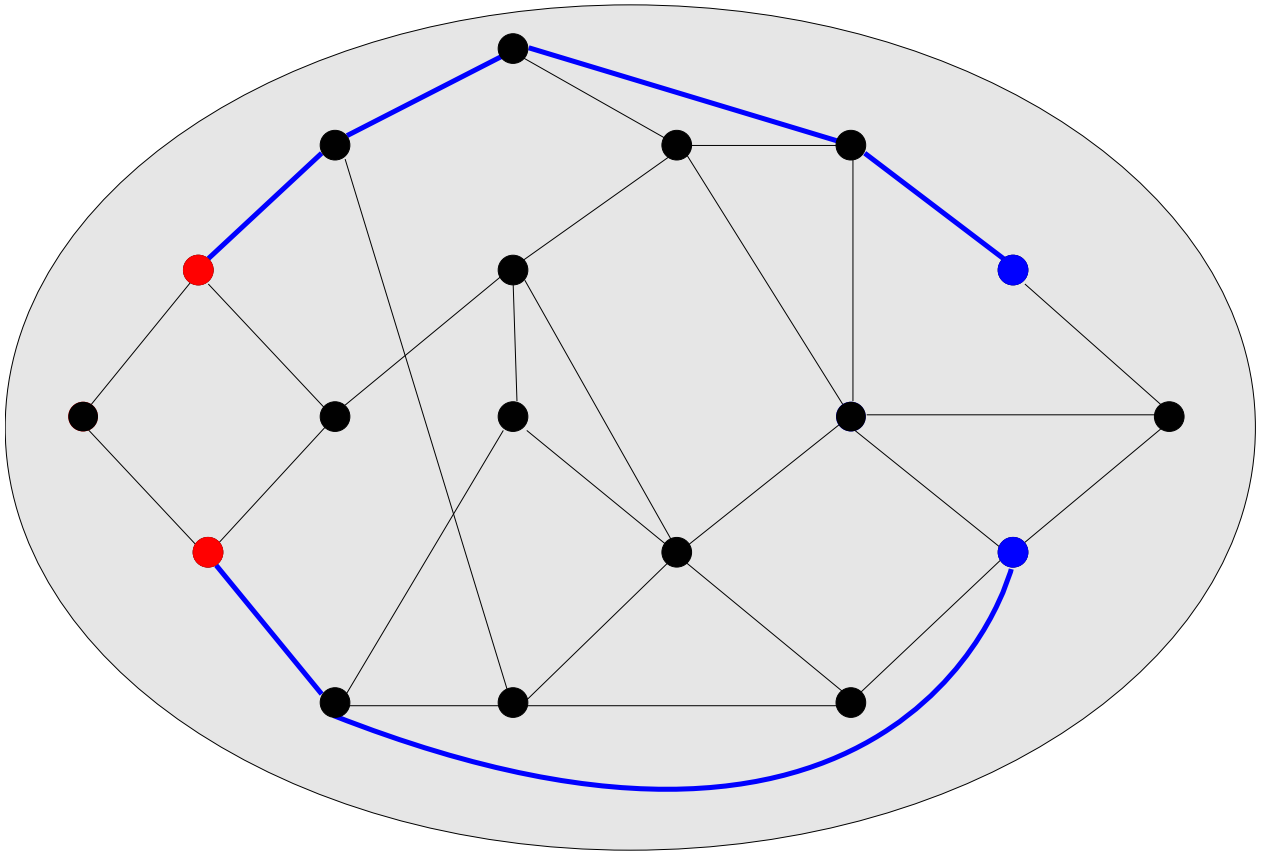


No efficient solution known (and conjectured not to exist)

Problem: $X-Y$ disjoint paths

$X-Y$-DISJOINT PATHS

Given a graph, two sets X, Y of vertices and $k \in \mathbb{N}$, find k *disjoint paths* between vertices in X and Y.

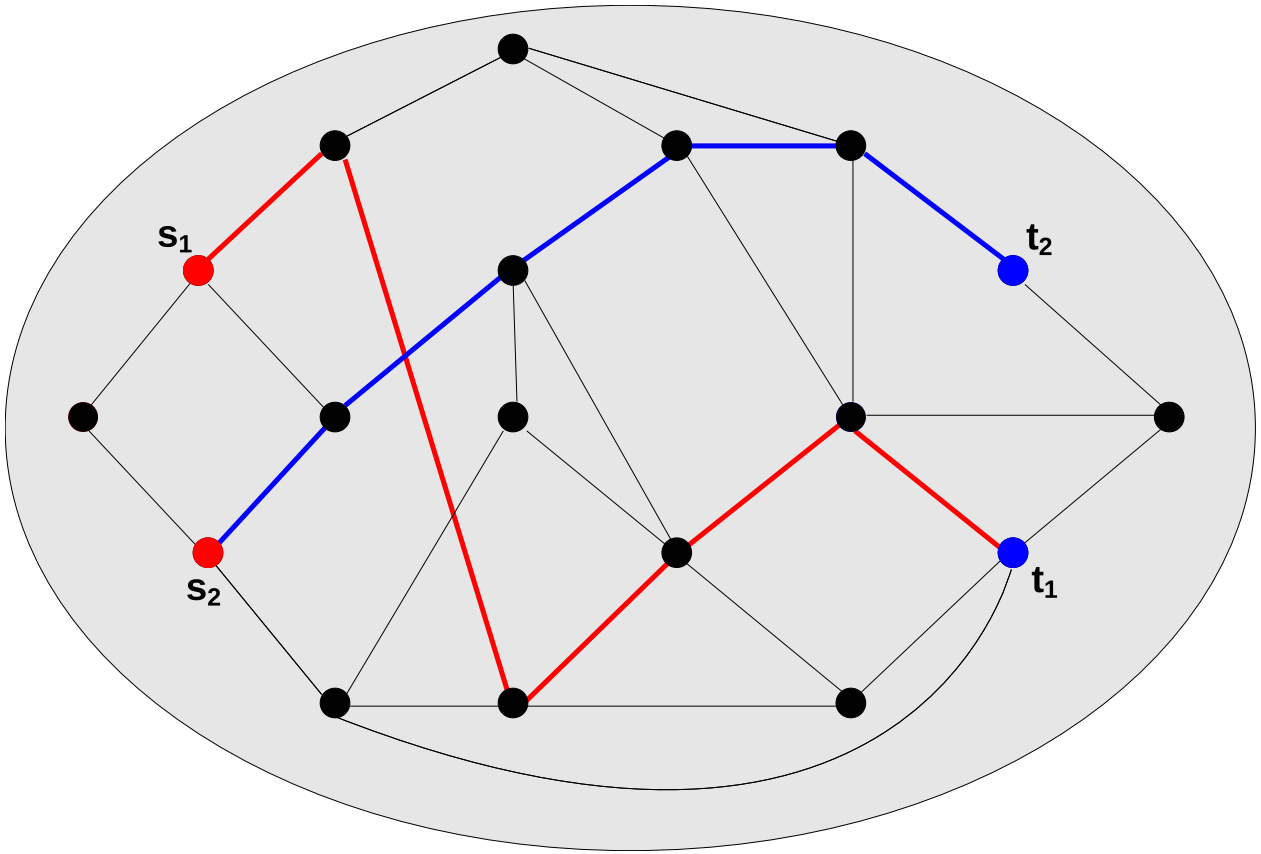


Can be solved efficiently using network flow techniques

Problem: disjoint paths

DISJOINT PATHS

Given a graph, two tuples $X := (s_1, \ldots, s_k)$, $Y := (t_1, \ldots, t_k)$ of vertices, find *disjoint paths* linking s_i, t_i, for all i.

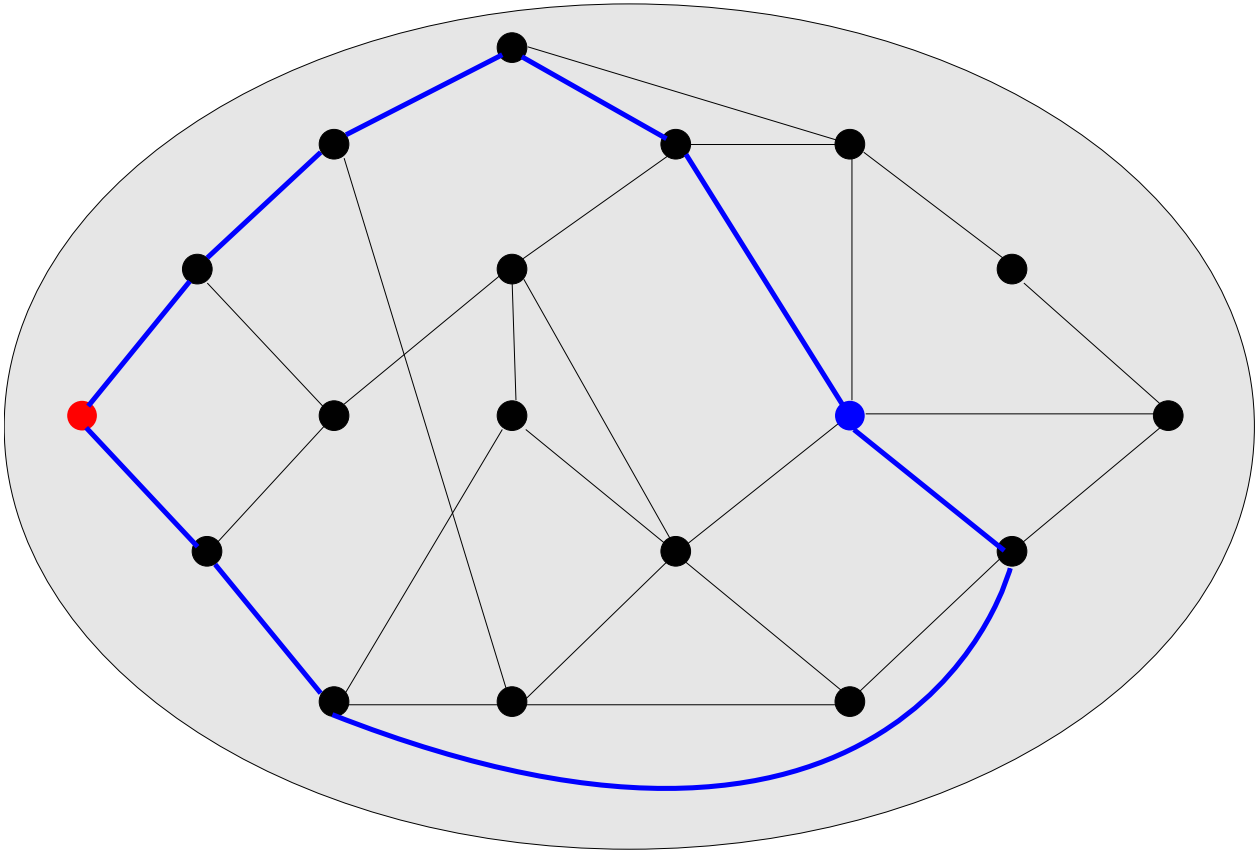


No efficient solution known (and conjectured not to exist)

Problem: Length Constrained Disjoint Paths

LENGTH CONSTRAINED DISJOINT PATHS

Given a graph, two vertices s, t and $c, k \in \mathbb{N}$, find k *disjoint paths* between s and t of length $\leq c$.



No efficient solution known (and conjectured not to exist)

Problem: Solving Linear Equations

INTEGER PROGRAMMING

Input: a system of linear equations

Problem: check whether it has an integer solution.

Example:

$$\begin{aligned} x + y &= 2 \\ y - 3z &= 5 \end{aligned}$$

No efficient solution known (and conjectured not to exist)

Problem: solving polynomial equations over integers

DIOPHANTINE EQUATIONS

Input: a system of Diophantine equations

Problem: check whether it has an integer solution.

Example:

$$\begin{aligned} xyz - y^3 + z^2 &= 2 \\ y - 3z &= 5 \end{aligned}$$

Undecidable — no algorithmic solution!

`https://en.wikipedia.org/wiki/Diophantine_equation`
`https://en.wikipedia.org/wiki/Hilbert's_tenth_problem`

Motivation

Questions.

- Why are some problems so much harder to solve than other – seemingly very similar – problems?
- Are they really harder to solve?

  Or have we just not found the right method to do so?

Computational Complexity:

classify problems according to the amount of resources (runtime, space, communication, etc) needed for their computation.

- Classify problems into classes of problems which are of the same "difficulty".
- Provide methods to establish the complexity of a problem.

Graph Problems: CLIQUE

Clique: A *clique* in a graph G is a complete subgraph of G.

MAX CLIQUE

Input: Graph G

Problem: Find largest clique $C \subseteq G$

That's an optimisation problem. Corresponding decision problem would specify a number k and ask for a "k-clique".

`https://en.wikipedia.org/wiki/Clique_problem`

...another notoriously hard problem; has an obvious brute-force algorithm.

CLIQUE

Can search for a k-clique of an n-vertex graph in time $O(n^k.k^2)$, poly(n) if k is constant.

Follow-up question:
Can we search for a k-clique in time $f(k).p(n)$, where $p(\cdot)$ is a polynomial?
Unlikely — CLIQUE is "fixed parameter intractable".

General observations

Classification of problems relies on various mathematical conjectures of which the most famous is the "P$\neq$NP" belief.

There are other, mostly stronger, ones, e.g. the "exponential time hypothesis", which has been used to prove that k-CLIQUE cannot be solved in time $n^{o(k)}$.

Lower bounds on runtime requirements are hard to show! Needs details of model of computation. More progress is often possible for lower bounds on *query complexity* and *communication complexity* of various problems.