Aalto University
School of Science

CS-E4530 Computational Complexity Theory

Lecture 1: Problems, Algorithms, Complexity, Reductions

Aalto University
School of Science
Department of Computer Science

Spring 2019

Agenda

- Problems vs. algorithms
- Efficiency of algorithms, complexity of problems
- Rates of growth
- Example problems
- Reductions between problems

Problems vs. Algorithms

This course focuses on analysing the *computational complexity of problems* (not algorithms).

- A *problem*: an infinite set of *instances* (possible inputs) with an associated *question*
- A *decision problem*: a question with a yes/no answer

Definition

REACHABILITY:
INSTANCE: A graph (V, E) and vertices $v, u \in V$.
QUESTION: Is there a path in the graph from v to u?

Before turning to the question of complexity of *problems* we consider the computational complexity of algorithms.

Algorithm for REACHABILITY

$S := \{v\}$; mark v;
while $S \neq \{\}$ **do**
    choose a vertex i and remove it from S;
    for all $(i,j) \in E$ **do**
        if j is not marked **then** mark j and add it to S
        endif
    endfor
endwhile ;
if u marked **then** return 'there is a path from v to u'
else return 'there is no path from v to u'
endif

How efficient is the algorithm?

- How is the efficiency affected by
  - Programming language?
  - Computer architecture?
  - Representation of the graph?
  - Representation of the set S?
- Typically the efficiency of an algorithm is measured by the *rate of growth* of the running time (or memory usage), i.e., by considering how the run time (or memory usage) of the algorithm *scales* when the size of the input increases.
- For this the O-notation is used where multiplicative and additive constants are ignored.
- For example, given some assumptions about the efficiency of the bookkeeping, the algorithm above terminates in $O(|E|)$ steps, where $|E|$ is the size (number of edges) of the graph.

Rates of Growth

Definition (The O-Notation)

Let f, g: $\mathbf{N} \mapsto \mathbf{R}^+$.

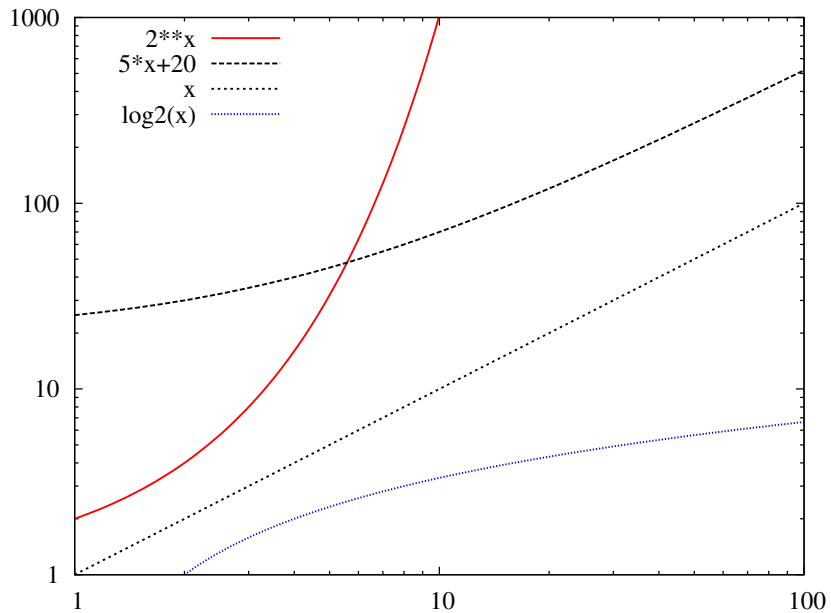
- $f(n) = O(g(n))$ (f *grows at most as fast as* g), if there exist $c > 0$ and $n_0 > 0$ such that for all $n \geq n_0, f(n) \leq c \cdot g(n)$
- $f(n) = \Theta(g(n))$, (f *grows exactly as fast as* g), if $g(n) = O(f(n))$ and $f(n) = O(g(n))$.
- $f(n) = o(g(n))$ (f *grows strictly slower than* g), if for any $c > 0$ there is an $n_c > 0$ such that for all $n \geq n_c, f(n) < c \cdot g(n)$
- $f(n) = \Omega(g(n))$, if $f(n) \neq o(g(n))$, i.e. if for some $c > 0$ there is for any $n > 0$ an $n' \geq n$ such that $f(n') \geq c \cdot g(n')$ (in other words "f *grows infinitely often at least as fast as* g")

Rates of Growth: Some Example Functions

Example

- If $p(n)$ is a polynomial of degree d, then $p(n) = \Theta(n^d)$.
- If $c > 1$ and $p(n)$ is a polynomial, then $p(n) = o(c^n)$, i.e.

  any polynomial grows strictly slower than any exponential.
- If $c \geq 0$, then $\log^c n = o(n)$.



Observe the logarithmic scales

Computational Complexity of Problems

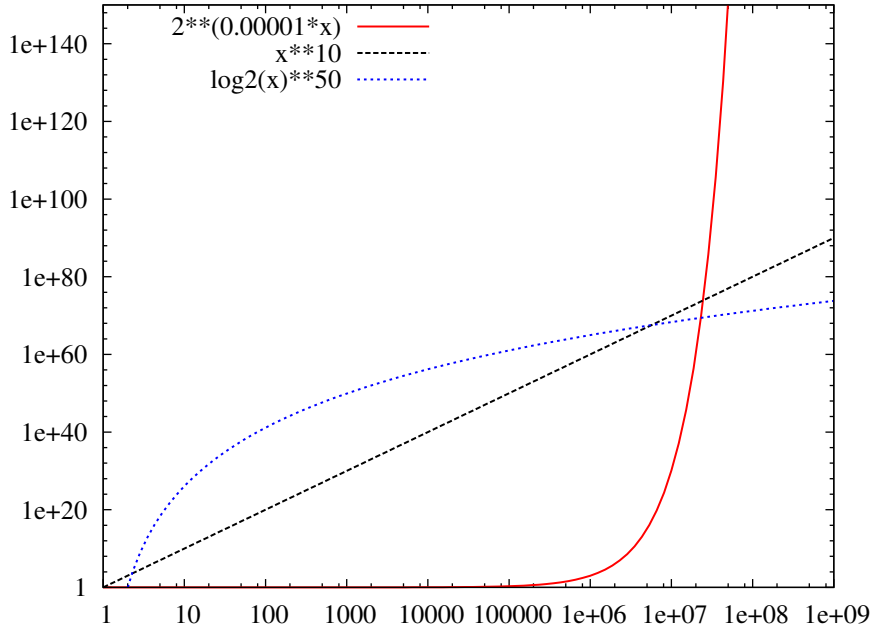
The following simplifying assumptions are typically made in considering the computational complexity of problems:

- A problem is *efficiently solvable* if there is an algorithm solving the problem, for which the rate of growth of the solution time is *polynomial* w.r.t. the size n of the input (i.e. $O(n^d)$ for some $d \geq 0$).
- A problem is *intractable* when no polynomial time algorithm is available for it.
- One assesses the *worst-case performance* (not e.g. average case).
- As the standard mathematical model of algorithms one uses *Turing machines*.

Discussion

Possible criticism:

- Not all polynomial time algorithms are efficient in practice.
  There are efficient computations that are not polynomial.
  For instance, consider n^{10} vs $2^{\frac{n}{100.000}}$.
- Average case analysis is more informative than worst-case. (In case one knows the distribution of inputs.)



☞ "*Adopting polynomial time worst-case performance as our criterion of efficiency results in an elegant and useful theory that says something meaningful about practical computation, and would be impossible without this simplification.*" (C. Papadimitriou 1994)

Some Example Problems

- Maximum flow
- Bipartite matching
- The travelling salesperson problem

Maximum Flow

Definition

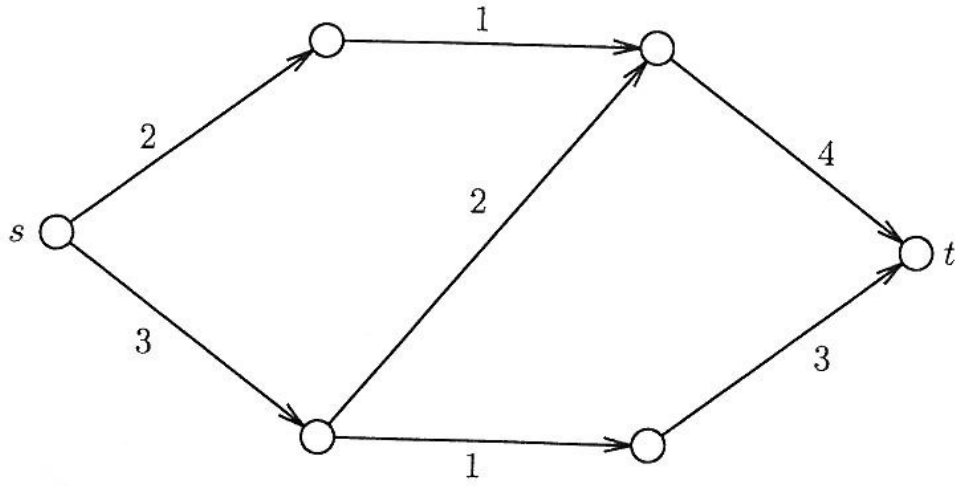
MAX FLOW
INSTANCE: Network $N = (V, E, s, t, c)$, where (V, E) is a (directed) graph, $s, t \in V$, the *source* s has no incoming edges, the *sink* t has no outgoing edges and c is a function giving a *capacity* for each edge (each $c(i,j)$ is a positive integer).
QUESTION: What is the largest possible value for the flow in N?

where

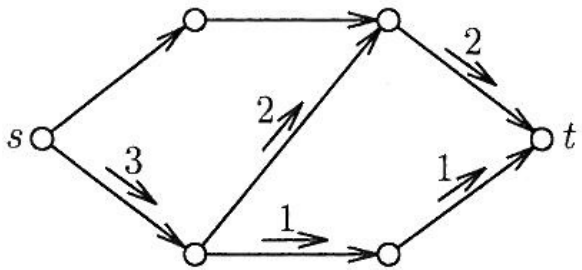
- A *flow* is a function f that assigns to each edge (i,j) a nonnegative integer $f(i,j) \leq c(i,j)$ such that for each vertex (except s and t) the sum of f's on the incoming edges is equal to the sum of f's on the outgoing edges.
- The *value* of a flow is the sum of the flows on edges leaving s.

Maximum Flow: An Example

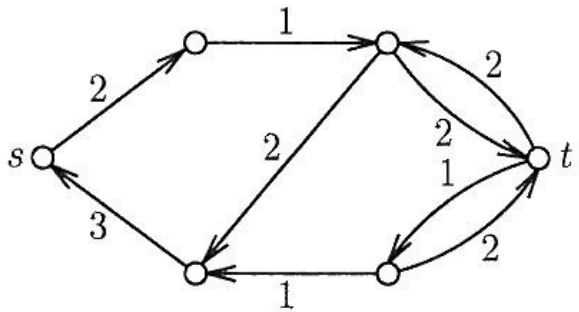
Network N:



Network flow f of value 3



f optimal iff t is not *reachable* from s in the augmentation network $N(f)$:



[Papadimitriou, 1994]

Maximum Flow: Discussion

- MAX FLOW is an *optimisation problem*.
- MAX FLOW(D) (decision problem)
  INSTANCE: Network N and integer K (goal/target value)
  QUESTION: Is there a flow of value K or more in N?
- MAX FLOW is an illustrative example of a problem where the challenge was for some time to find a polynomial time solution method.
- When the "*barrier of exponentiality*" was broken, more and more efficient polynomial time algorithms were developed $(O(n^5), \ldots, O(n^3), \ldots)$

Bipartite Matching

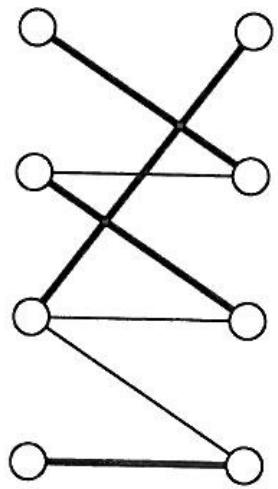
Definition

MATCHING

INSTANCE: Bipartite graph $G = (U, V, E)$, where $U = \{u_1, \ldots, u_n\}$, $V = \{v_1, \ldots, v_n\}$, and $E \subseteq U \times V$.

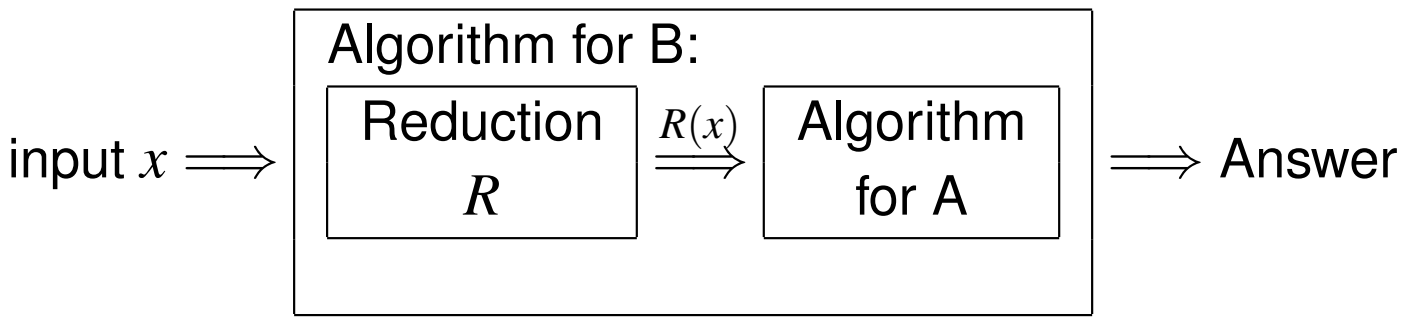
QUESTION: Is there a set $M \subseteq E$ of n edges such that for any two edges $(u, v), (u', v') \in M$, $u \neq u'$ and $v \neq v'$

(i.e., is there a *perfect matching*)?



Reductions Between Problems

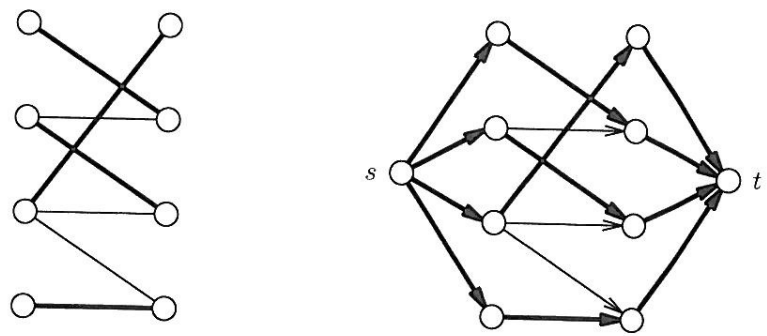
- A *reduction* from a problem B to a problem A is an algorithm R that transforms any instance x of B to an "equivalent" instance $y = R(x)$ of A.
- If the reduction R is efficient, and an efficient algorithm for A exists, then the two can be combined to yield an efficient algorithm for B.



- In a sense, R transforms problem B into a *special case* of problem A.

Reductions: An Example

- MATCHING can be solved by a *reduction* to MAX FLOW:
  Given any bipartite graph $G = (U, V, E)$, construct a network
  $$N = (U \cup V \cup \{s, t\}, E', s, t, c),$$
  where $E' = E \cup \{(s, u) \mid u \in U\} \cup \{(v, t) \mid v \in V\}$
  and all capacities are equal to 1.
- Now G has a perfect matching iff N has a flow of value n. (The claim from right to left follows from the so called Ford-Fulkerson integral flows theorem.)



[Papadimitriou, 1994]

The Travelling Salesperson Problem

Definition

TSP

INSTANCE: n cities $1, \ldots, n$ and a nonnegative integer distance d_{ij} between any two cities i and j (such that $d_{ij} = d_{ji}$).

QUESTION: What is the shortest tour of the cities, i.e., a permutation π such that

$$\sum_{i=1}^{n} d_{\pi(i)\pi(i+1)}$$

is as small as possible (where $\pi(n+1) = \pi(1)$).

Decision problem TSP(D): is there a tour of length at most B (budget)?

The Travelling Salesperson Problem: Discussion

- A naive algorithm for TSP: enumerate all possible permutations, compute the cost of each, and pick the best.
  Not very practical: $O(n!)$ tours.
- Consider e.g. a 25-city TSP instance:
  - $n = 25 \Rightarrow n! \approx 1.6 \times 10^{25}$
  - 1 route/μs $\Rightarrow$ runtime $\approx$ 500 bn years
  - age of the universe $\approx$ 10-20 bn years
- For TSP no polynomial-time algorithm is known, despite decades of intensive efforts at developing one.
- Conjecture: there can be no polynomial-time algorithm for TSP.
- This is closely related to one of the most important open problems in computer science: is P $=$ NP?[1]

[1] This is one of the seven "millennium" mathematical problems defined by the Clay Mathematics Institute in the year 2000 (`http://www.claymath.org/`). The first person resolving this question will be awarded 1.000.000 USD by the Institute.

Learning Objectives

- Ability to read and formulate decision/optimisation problems
- Basic understanding of growth rates (polynomial vs. exponential)
- The idea of reducing one problem to another