1 Introduction to Fine-Grained Complexity Theory

Version: 180

1.1 Overview over the course

Motivation

Suppose you are an algorithms & complexity theory researcher who is approached by a practitioner, desperate to solve problem A. How can you help her? If you manage to come up with a **polynomial-time algorithm**, this would be a good starting point for her. However, if you fail to find a polynomial-time algorithm, but can prove A to be NP-**hard**, this is still a good outcome for her: it is likely to convince her to *relax the problem*[1] to a new formulation B instead of solving A.

[1] She could, e.g., find a more specific formulation, or resort to **approximation**, or try to identify small input parameters (cf. **fixed-parameter tractability**), or she might feel justified to use **heuristics**.

Now suppose you indeed manage to find a polynomial-time algorithm for the relaxed problem B – say, it runs in quadratic time –, but it performs infeasibly slow on her input data (think of modern, BIG data applications). Could there still be a fast, e.g., **(near-)linear time algorithm**? Obviously, to rule out this possibility, you cannot use NP-hardness (after all, your quadratic-time algorithm proves A to be in P). But if you neither manage to find a (near-)linear time algorithm, nor can come up with a justification for this, both of you will be unhappy: She receives no advice what to do (should she relax the problem? Should she try harder to find a fast algorithm?) and you will be embarrassed that you could not help her.

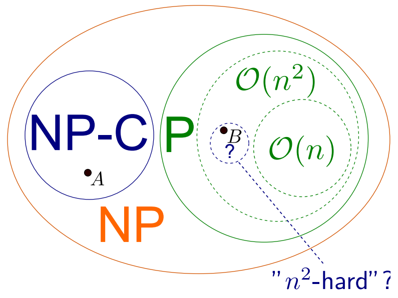


Figure 1.1: Is there an analogue for the class of NP-complete problems **inside** P?

The question arises: can we find tools to distinguish whether A has only algorithms with a high polynomial running time like $\mathcal{O}(n^c)$ with a large c or whether it admits a solution running in near-linear time $\tilde{\mathcal{O}}(n)$?

Notation: $\tilde{\mathcal{O}}(T)$ denotes a running time $T\text{polylog}T$.

Can we find an analogue of NP*-hardness* ***within*** P*?*

THIS COURSE: We study barriers for improving over specific running times. In particular, we investigate approaches how to prove, given some problem of interest B, lower bounds for the best possible polynomial running time $\mathcal{O}(n^c)$ for B – note that these will always rely on some plausible **hardness assumption**. Along the way, we will learn advanced algorithmic tools that give the fastest currently known algorithms for fundamental problems in this field.

Methods

OUR GENERAL APPROACH to establish fine-grained hardness results is to prove **conditional lower bounds**[2]: Consider a problem of interest B for which the fastest algorithm we know runs in time $\mathcal{O}(n^d)$. To explain its hardness, we take another problem C, solvable in some (other) running time $\mathcal{O}(n^c)$. We should be reasonably confident to conjecture that C cannot be solved significantly faster, i.e., that no $\mathcal{O}(n^{c-\varepsilon})$-time algorithm exists for any constant $\varepsilon > 0$. We then relate C to B in such a way that an $\mathcal{O}(n^{d-\varepsilon})$-time algorithm for B (for any $\varepsilon > 0$) would give an $\mathcal{O}(n^{c-\varepsilon'})$-time algorithm for C (for some $\varepsilon' > 0$).[3] Thus, if the conjecture for C is true, we obtain a tight lower bound for problem B.

[2] Why don't we prove **unconditional** lower bounds? The simple reason is: We don't know how. E.g., current techniques cannot even show that 3-SAT has no *quadratic* time algorithms.

[3] We give a formal definition of such **fine-grained reductions** later in the course. A first example is given below.

Fundamental problems

Let us give some examples for fundamental problems that can be used to derive conditional lower bounds (these are candidates for taking the role of problem C). The list below gives the main conjectures that we will use throughout this course.

Problem 1.1.
All-Pairs-Shortest-Paths (APSP)

Given:	*A weighted graph G on n nodes.*
Determine:	*The distance between any pair of nodes u, v in G.*
Conjecture:	*No $\mathcal{O}(n^{3-\varepsilon})$-time algorithm exists.*

Problem 1.2.
3-SUM

Given:	*A list of n integers $a_1, \ldots, a_n$.*
Determine:	*Is there a triplet i, j, k such that $a_i + a_j + a_k = 0$?*
Conjecture:	*No $\mathcal{O}(n^{2-\varepsilon})$-time algorithm exists.*

Problem 1.3.

CNF-Satisfiability (CNF-SAT)

Given: *A Boolean formula ϕ in conjunctive normal form (CNF), with N variables and M clauses.*

Determine: *Is there a satisfying assignment for ϕ?*

Conjecture: *No $\mathcal{O}((2-\varepsilon)^N \mathrm{poly}(M))$-time algorithm exists.*

CNF-SAT is actually an example with a conjectured *exponential*-time lower bound. Interestingly, we will see that it also explains a number of hardness results in the *polynomial*-time regime.

Problem 1.4.

Orthogonal Vectors (OV)

Given: *sets $A, B \subseteq \{0,1\}^d$ of size n*

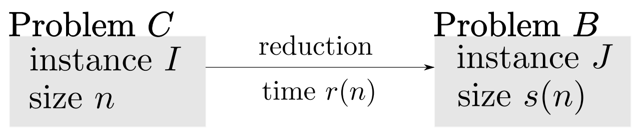
Determine: *Is there an* ***orthogonal pair*** *$a \in A, b \in B$?*

Conjecture: *No $\mathcal{O}(n^{2-\varepsilon} \mathrm{poly}(d))$-time algorithm exists.*

The orthogonal vectors problem has a close connection with CNF-SAT, which we highlight in an upcoming lecture.

1.2 Our types of reductions

How do we relate problems C and B to each other? The simplest type of such a **fine-grained reduction** is the following:

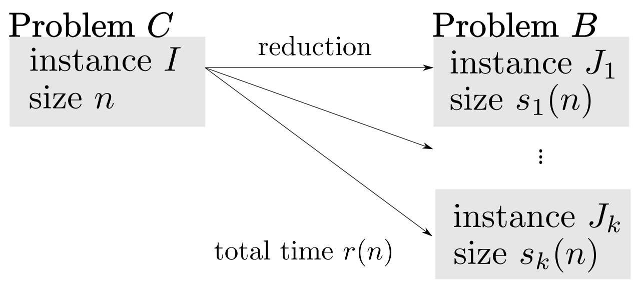


Such a reduction maps, in time $r(n)$, any size-n instance I of problem C to an *equivalent* size-$s(n)$ instance J of problem B, i.e.,

$$I \text{ is a YES-instance for } C \iff J \text{ is a YES instance for } B.$$

By this reduction, any $T(n)$-time algorithm for B yields a $\mathcal{O}(r(n) + T(s(n)))$-time algorithm for C: we first run the reduction to produce J and then use the $T(n)$-time algorithm for B to solve J. Thus, if some conjecture rules out an $\mathcal{O}(r(n) + T(s(n)))$-time algorithm for C, we obtain the conditional lower bound that no $T(n)$-time algorithm for B exists (under this conjecture).

We will also regard reductions mapping to multiple instances of B:



Here, the reduction may use the answers to instances $J_1, \ldots, J_k$ in an arbitrary way to determine the answer for instance I of C quickly. By such a reduction, similar to above, any $T(n)$-time algorithm for B yields a $\mathcal{O}(r(n) + \sum_{i=1}^{k} T(s_i(n)))$-time algorithm for C, which again may be used to derive a conditional lower bound.

1.3 *Some Conditional Lower Bounds*

A classic textbook example for dynamic programming is the **longest common subsequence (LCS)** problem. For strings of length n, it yields a simple $\mathcal{O}(n^2)$-time algorithm. Surprisingly, substantial improvements over this simple approach are unlikely: Assuming the conjecture for OV (or CNF-SAT), we will prove that there is no $\mathcal{O}(n^{2-\varepsilon})$-time algorithm.

p **a** s s e **n g e r**
m **a n** a **g e r**

Figure 1.2: Longest Common Subsequence example.

Another example is the following: Suppose you are given a $n \times n$ matrix with integer entries. Your aim is to choose a submatrix (spanned by consecutive rows and coloumns) with the maximum sum of weights. It is not too difficult to see that you can solve this problem in time $\mathcal{O}(n^3)$ ($\rightarrow$ **exc.**). Surprisingly, as we shall see later in the course, a substantially faster algorithm (running in time $\mathcal{O}(n^{3-\varepsilon})$) exists **if and only if** a substantially faster algorithm exists for APSP!

We will encounter the 3SUM conjecture as particularly prevalent in **computational geometry**. Here, it can be shown to imply quadratic-time lower bounds for problems such as determining whether in a set of n points in the plane, we find a triplet of points lying on a common line.

Finally, consider **Subset Sum**: Given a set S of n non-negative integers and a target t, the task is to determine whether some subset of S sums up to t. Until recently, the state of the art used to be an $\mathcal{O}(nt)$ dynamic programming algorithm. Notably, recent results in fine-grained complexity give both an $\tilde{\mathcal{O}}(t)$-time algorithm and a matching lower bound of $t^{1-o(1)}$ (based on the complexity of satisfiability).

In Figure 1.3, we give a glimpse (far from comprehensive!) into the web of reductions that is currently emerging within P.

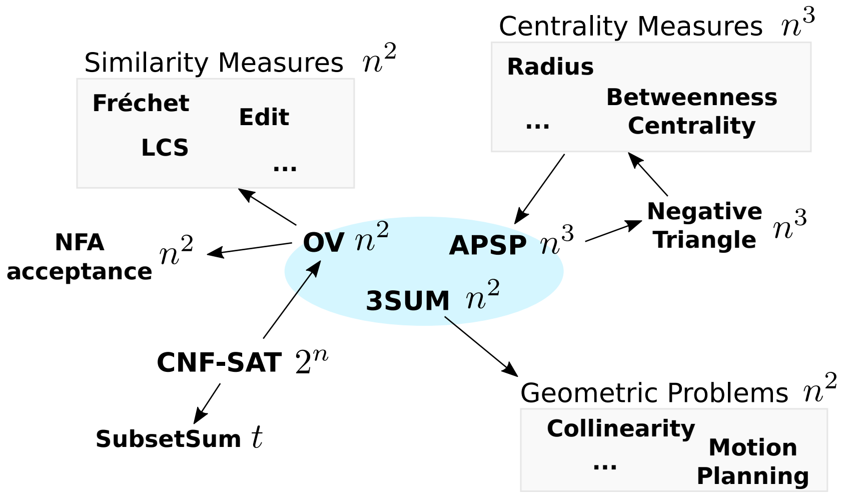


Figure 1.3: A (partial) web of reductions: Conditional lower bounds from OV, APSP, 3SUM (and SAT).

1.4 *A first example: OV and NFA acceptance*

We introduce one of the most important problems for the hardness of quadratic-time problems.

$a, b \in \{0,1\}^d$ are **orthogonal**
iff $\langle a, b \rangle = 0$
iff $\sum_{k=1}^{d} a[k] \cdot b[k] = 0$
iff $\forall k \in [d]$: $a[k] = 0$ or $b[k] = 0$.
Notation: Here, $[d] := \{1, \dots, d\}$, and $a[k]$ refers to the k-th coordinate of vector a.

Problem 1.5.
Orthogonal Vectors (OV)
Given: *sets $A, B \subseteq \{0,1\}^d$ of size n*
Determine: *Is there an* ***orthogonal pair*** *$a \in A, b \in B$?*

Baseline Algorithms:

- trivial enumeration of pairs: $\mathcal{O}(n^2 d)$
- slightly more difficult: $\mathcal{O}(nd2^d)$ ($\rightarrow$ **exc.**)

These algorithms do not violate the following hypothesis.

Hypothesis 1.6 (OV-Hypothesis (OVH))**.** *For no $\varepsilon > 0$, there exists an $\mathcal{O}(n^{2-\varepsilon}\text{poly}(d))$-time algorithm for OV.*

Outlook: We will discuss what range of dimensions make OV "hard" in more detail in future chapters.

Influence of d: We are particularly interested in the regime of d for which the known OV algorithms have a **quadratic running time** in the input:

1. $d = n^{o(1)}$: Thus, the input size is $2nd = \mathcal{O}(nd) = n^{1+o(1)}$. (Otherwise, the first algorithm is faster than quadratic in $2nd$)
2. $d = \omega(\log n)$: If, e.g., $d = \frac{1}{2}\log n$, then the second algorithm solves OV in subquadratic time $\mathcal{O}(n^{\frac{3}{2}})$.

   A helpful regime to think of is $d = \Theta(\log^2 n)$.

We will show that OV implies a quadratic time lower bound for NFA acceptance problem. Recall that a nondeterministic finite automaton (NFA) M is a directed graph with a designated initial node s and a set of accepting nodes T, where each edges receive a label from an alphabet Σ. We say that M accepts a string x over Σ if there is a walk in the graph from s to some accepting node $t \in T$ such that the i-th edge in the walk is labelled with $x[i]$.

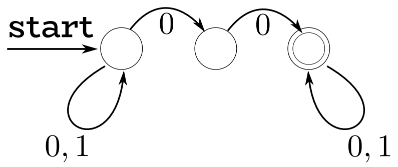


Figure 1.4: Example of an NFA over $\Sigma = \{0,1\}$, accepting bit strings with at least two consecutive 0's. The accepting state is indicated by double circles. Duplicate edges with different labels are displayed as a single edge with multiple labels.

Problem 1.7.

NFA acceptance

Given:	*NFA M, string x*
Determine:	*Does M accept x?*

ALGORITHM: NFA acceptance is solvable in time $\mathcal{O}(|M| \cdot |x|)$, where $|M|$ denotes the size[4] of M and $|x|$ denotes the length of x. To see this, use dynamic programming: compute a table $T[i]$ with $i \in [|x|]$ as follows:

[4] The size of M is the number of its states plus transitions.

$T[i]$ – set of states reachable in M when reading the prefix $x[1..i]$

By letting $T[0]$ consist of the initial node(s) of M, we can compute $T[i]$ from $T[i-1]$ by determining all states q such that some $q' \in T[i-1]$ has a transition from q' to q labelled $x[i]$. This can be done in time $\mathcal{O}(|M|)$ per i.

Given the simplicity of this algorithm, you might ask whether some faster algorithm exists. Surprisingly, no polynomially faster algorithm exists if the OV hypothesis is true.

Theorem 1.8 (Impagliazzo)**.** *If the OV hypothesis is true, then for no $\varepsilon > 0$, NFA acceptance can be solved in time $\mathcal{O}((|x| \cdot |M|)^{1-\varepsilon})$.*

Proof. We give a linear-time reduction that takes an OV instance A, B of n vectors in $\{0,1\}^d$ and produces an equivalent NFA acceptance instance with an NFA M of size $\mathcal{O}(nd)$ and a string x of length $\mathcal{O}(nd)$. Pictorially, we depict such a reduction as follows:

OV

- n vectors
- d dimensions

$\xrightarrow{\text{time } \mathcal{O}(nd)}$

NFA acceptance

- NFA size $\mathcal{O}(nd)$
- string length $\mathcal{O}(nd)$

Such a reduction shows that an $\mathcal{O}((|M| \cdot |x|)^{1-\varepsilon})$-time algorithm for NFA acceptance would give an OV algorithm that runs in time

$$\mathcal{O}(\ ((nd)(nd))^{1-\varepsilon}\ ) = \mathcal{O}(n^{2-2\varepsilon} \cdot d^{2-2\varepsilon}) = \mathcal{O}(n^{2-\varepsilon'} \mathrm{poly}(d)),$$

where $\varepsilon' = 2\varepsilon$. This would refute the OV hypothesis!

We develop the reduction as specified above in two steps.

Step 1: Vector Orthogonality For any vector $a \in \{0,1\}^d$, we define a NFA $M(a)$ representing a as follows:

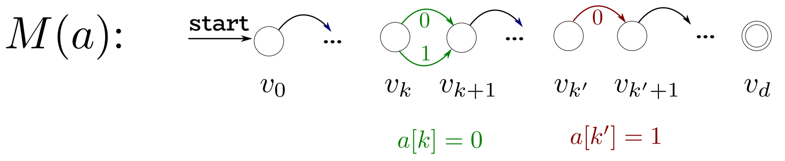


Figure 1.5: Vector NFA construction.

The following claim follows immediately from the construction of $M(a)$.

Claim 1.9. *For any $b \in \{0,1\}^d$, $M(a)$ accepts the string $b[1]\,b[2]\dots b[d]$ if and only if a, b are orthogonal.*

Step 2: Final string construction Given the vector sets $A = \{a_1, \dots, a_n\}$, $B = \{b_1, \dots, b_n\}$, we construct the following machine M representing all vectors in A as in Figure 4

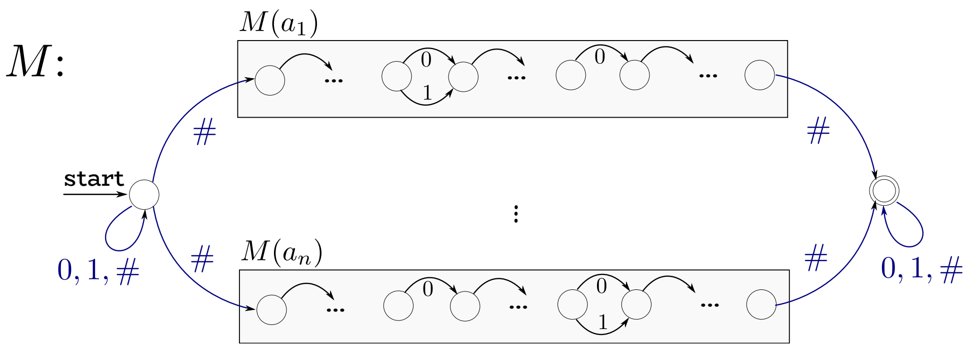


Figure 1.6: Final NFA construction.

Finally, we define the string x as

$$x := \#\, b_1[1]\dots b_1[d]\, \#\, b_2[1]\dots b_2[d]\, \#\ \dots\ \#\, b_n[1]\dots b_n[d]\, \#.$$

Claim 1.10. *M accepts x if and only if there is an orthogonal pair $a_i \in A$, $b_j \in B$.*

Indeed, for any orthogonal pair a_i, b_j, an accepting path in M is given by staying in the initial state for the prefix

$$\#b_1[1]\dots b_1[d]\#\dots\#b_{j-1}[1]\dots b_{j-1}[d],$$

then using the #-transition to the initial state of $M(a_i)$, traversing it (which is possible by Claim 1.9 of a, b) until the accepting state of

$M(a_i)$, and completing the traversal of M by #-traversing to the accepting state and staying there for the remaining suffix of x.

Conversely, it is easy to verify that any accepting path in M must traverse some $M(a_i)$ from its initial to its accepting state. By Claim 1.9, this requires that some substring of x represents an orthogonal vector to a_i. By construction of x, this is only possible if B contains a vector b_j that is orthogonal to a_i.

Note the previous claim shows that M and x yield an NFA acceptance instance that is equivalent to the OV instance A, B. By construction, M has size $\mathcal{O}(nd)$ and x consists of $\mathcal{O}(nd)$ characters. Furthermore, M and x can be computed in time $\mathcal{O}(|M| + |x|) = \mathcal{O}(nd)$. Thus, as proven above, an $\mathcal{O}((|M| \cdot |x|)^{1-\varepsilon})$-time NFA acceptance algorithm would refute the OV hypothesis. $\square$

1.5 *Machine model*

A big advantage of complexity classes like P and NP is that their definition is robust against the choice of the computational model: Any computational model that is polynomial-time equivalent to a single-tape deterministic Turing machine gives rise to the same classes. For our fine-grained questions, however, the polynomial-time differences matter. Thus, we need to be more careful in our choice of computational model.[5]

[5] As an example, single-tape Turing machines require quadratic time to detect palindromes – a task that we would assume to be solved by any realistic model of computation in linear time.

OUR MODEL is the *random-access machine (RAM)*. Its storage is divided into *cells*. The content of a cell is sometimes called a *word*. It is assumed that each cell can store a $\Theta(\log n)$-bit number. The CPU may store a constant number of words. It can perform all "standard" operations on one or two words (i.e., arbitrary arithmetic or logical operations) in time $\mathcal{O}(1)$. Here, it has random access to all words.

1.6 *Advantages and Drawbacks*

As illustrated above, to obtain **fine-grained running time lower bounds** (in particular, to classify polynomial-time problems), conditional lower bounds provide a useful tool. Analogously to NP-hardness, they give convincing criteria to abandon the search for a faster algorithm: E.g., before searching for a subquadratic NFA acceptance algorithm, you might as well first try to refute the conjecture for OV – if you don't think you have the necessary tools for this, then you should give up on NFA acceptance and rather relax the problem or regard alternatives instead.

What if the conjectures turn out to be false? Aren't we relying crucially on these conjectures? Interestingly, several advantages of the above approach are somewhat independent of whether these conjectures eventually turn out to be true or false:

1. Even if a conjecture C eventually turns out to be false, a conditional lower bound based on C still shows that any algorithm breaking this lower bound needs *techniques that are strong enough to break* C. In particular, if we believe that C is a barrier for current techniques, or if we search for algorithms that avoid an "algorithmic hammer" that seems necessary to break C, these lower bounds are highly informative.
2. While a conjecture might be refuted, the uncovered relationship persists, potentially giving further insights. E.g., even if the best algorithm for OV would be subquadratic, but still superlinear, say $\mathcal{O}(n^c \text{poly}(d))$ with $1 < c < 2$, then NFA acceptance would not have an algorithm running in near-linear time $\tilde{\mathcal{O}}(|M| + |x|)$.
3. Sometimes, the lack of (higher) conditional lower bounds *inspires algorithmic improvements*! There are examples in the literature where the quest for tight lower bounds in fact suggested a route to obtain faster algorithms.
4. A web of conditional lower bounds structures the search for faster algorithms. It seems more likely to be successful if we attack the conjecture on which a lower bound is based, rather than the lower bound itself.

We (Karl and Marvin) remain agnostic about whether the above conjectures for OV, 3SUM, APSP and CNF-SAT hold. Rather, we view them as important **hypotheses**: If they turn out to be true, they give a very detailed picture of the complexity of polynomial-time problems. But even if they are eventually refuted, conditional lower bounds based on them seem likely to reveal fundamental structures within P (and in fact, may even help in their refutation).

Terminology: From now on, we typically call our hardness assumptions *hypotheses* rather than *conjectures*, to stress that we do not fundamentally conjecture all of them to be correct.

1.7 *Structure of this course*

The main part of this course concerns our main hypotheses: the OV hypothesis and the Strong Exponential Time Hypothesis for SAT, the 3SUM hypothesis, and the APSP hypothesis. In each of these cases, we discuss both the fastest known algorithm for the underlying problem, as well as how to prove conditional lower bounds based on this assumption.

Later in the course, we turn towards (partial) relationships between these problems, non-standard hardness assumptions, hardness of approximation and other settings of computation (including nondeterminsm and dynamic algorithms).

The aim of this course is to learn (1) how to prove conditional lower bounds, (2) algorithmic techniques to obtain the fastest known algorithms for fundamental problems, and (3) an understanding of the complexity landscape of polynomial-time problems.