Advanced Algorithmics

Strategies for Tackling Hard Problems

Sebastian Wild

Markus Nebel

Lecture 1

2017-04-20

0 How to Solve Hard Problems?

View on NP-completeness in elementary courses:

"I can't find an efficient algorithm, but neither can all these famous people."

Garey, Johnson 1979

. . . but this is not the end of the story

"you had just neatly sidestepped potential charges of incompetence by proving that the bandersnatch problem is NP-complete.
However, the bandersnatch problem had ***refused to vanish at the sound of those mighty words****, and you were still faced with the task of finding some usable algorithm for dealing with it."*

Garey, Johnson 1979

Look for loopholes in theory

- pseudopolynomial algo
- parametrized complexity
- approximation algo
- randomized algos.
- smoothed analysis
- average-case analysis
- quantum computing ?

} provide no new algos.

1 P vs. NP revisited

1.1 Model of Computation

- actual hardware is finite
  - memory
  - input
  - time

many different possibilities to generalize

- In theory we need asymptotic struts.
  - abstract from irrelevant detail
  - make analysis feasible
  - allow conclusive comparison of different algor.

Extended Church-Turing Thesis

any realistic model of computation can be simulated with only polynomial overhead on a TM.

↳ in case of doubt → TM

Definition 1.1 (Time and Space Complexity)

Let Σ_I and Σ_O two alphabets and A an algorithm implementing a total mapping $\Sigma_I^\star \to \Sigma_O^\star$. Then for each $x \in \Sigma_I^\star$ we denote by $Time_A(x)$ (resp. $Space_A(x)$) the logarithmic time complexity (resp. logarithmic space complexity) for A on x. ◀

- uniform cost model $\qquad a = a^2$ too cheap
  - all arithmetic operations take $O(1)$ time
- logarithmic cost model
  - costs are proportional to size of numbers in bits

TM is always in logarithmic model

since Σ fixed

word-RAM

In practice we have two regimes

- "ints" suffices $O(1)$ suffices add, mult, bit tricks
- numbers are too large
  ↳ BigInteger store numbers as strings

unified: word size w = # bits in word
can depend on n

$w(n) = \Theta(\log n)$
↳ sort in $O(n \log \log n)$.

Definition 1.2 (Worst-Case Complexity)

Let Σ_I and Σ_O two alphabets and A an algorithm implementing a total mapping $\Sigma_I^\star \to \Sigma_O^\star$. The *worst case time complexity of* A is the function $Time_A : \mathbb{N} \to \mathbb{N}$ with

$$Time_A(n) = \max\{Time_A(x) \mid x \in \Sigma_I^n\},$$

for each $n \in \mathbb{N}$. The *worst case space complexity of* A is given by function $Space_A : \mathbb{N} \to \mathbb{N}$ with

$$Space_A(n) = \max\{Space_A(x) \mid x \in \Sigma_I^n\}.$$

◀

Definition 1.3 (Decision Problem and Algorithms)

A *decision problem* is given by $P = (L, U, \Sigma)$ for Σ an alphabet and $L \subseteq U \subseteq \Sigma^\star$. An algorithm A *solves (decides)* decision problem P, if for all $x \in U$

1. $A(x) = 1$ for $x \in L$, and
2. $A(x) = 0$ for $x \in U \setminus L$ (i.e. $x \notin L$)

holds. Here $A(x)$ denotes the output of A on input x. If $U = \Sigma^\star$ holds we denote P briefly by (L, Σ). ◀

$\rightsquigarrow$ A effectively computes a total function $A : U \to \{0, 1\}$.

Theorem 1.4 (Inconsistency of Complexities)
There is a decision problem $P = (L, \{0, 1\})$ such that for any algorithm A that solves P there is another algorithm B that also decides P but with

$$Time_B(n) \quad = \quad \log_2\big(Time_A(n)\big)$$

for an infinite number of natural numbers n. ◀

Definition 1.5 (Upper/Lower Bounds, Optimal Algorithms)
Let U be an algorithmic problem and f, g functions $\mathbb{N}_0 \to \mathbb{R}^+$.

- We call $\mathcal{O}(g(n))$ an *upper bound for time complexity of U* if an algorithm A exists that solves U in time $Time_A(n) \in \mathcal{O}(g(n))$.
- We say $\Omega(f(n))$ is a *lower bound for time complexity of U* if each algorithm B that solves U needs time $Time_B(n) \in \Omega(f(n))$.
- An algorithm C is called *optimal for U* if $Time_C(n) \in \mathcal{O}(g(n))$ and $\Omega(g(n))$ is a lower bound for the time complexity of U.

◀

Goals:

- formally specify what problems are practically solvable
- method to classify problems into tractable and intractable ones

1.2 The Classes P and NP

Definition 1.6 (P, tractable)
We define the class of languages $\mathcal{P}$ decidable in polynomial time by

$$\mathcal{P} = \Big\{ L = L(M) \;\Big|\; M \text{ is a Turing machine (algorithm)} \wedge Time_M(n) \in \mathcal{O}(n^c) \text{ for a } \underline{c} \in \mathbb{N} \Big\}.$$

A language (a decision problem) $L \in \mathcal{P}$ is called *tractable / efficiently decidable*. ◀

- robust against changes in computational model ECTT
- unless c is huge, $O(n^c)$-time algorithms usually practicle
- polynomials have the property that $n \rightsquigarrow 2n$ only gives a fixed factor more in time

How to show a problem in $\mathcal{P}$?

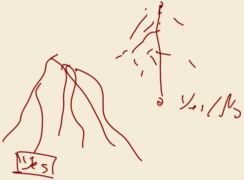
→ find a poly-time algorithm that solves the problems

How to show that it is not in $\mathcal{P}$?

Definition 1.7 (Nondeterminism, NP)

Let M a nondeterministic Turing machine (algorithm) and L a language over alphabet Σ.

- M accepts L ($L(M) = L$), for all $x \in L$ there is at least one computation of M which accepts x and for all $y \notin L$ every computation of M rejects y.
- For each $x \in L$ the *time complexity $Time_M(x)$ of M on x* is given by the time needed by a shortest accepting computation of M on x.
- The *time complexity of M* is a function $\mathbb{N} \to \mathbb{N}$ defined by $Time_M(n) = \max\{Time_M(x) \mid x \in L(M) \cap \Sigma^n\}$.



- We define class $\mathcal{NP}$ by

$$\mathcal{NP} = \{L(M) \mid M \text{ nondeterministic } TM \text{ with polynomial time}\}.$$

Non-determinism is unintuitive

Definition 1.8 (Certificates, Verifier, VP)

Let $L \subseteq \Sigma^\star$ a language.

- An algorithm A acting on inputs from $\Sigma^\star \times \{0,1\}^\star$ is called *verifier for L* (notation $L = V(A)$), if

$$L = \{w \in \Sigma^\star \mid (\exists c \in \{0,1\}^\star)(A(w,c) = \overset{\text{Yes}}{1})\}.$$

  Does A accepts input (w,c) we say c is proof (or certificate) for $w \in L$.

- A verifier A for L is a *polynomial-time verifier* if there is a $d \in \mathbb{N}$ such that for all $w \in L$, $Time_A(w,c) \in \mathcal{O}(|w|^d)$ for a proof c for $w \in L$.

- We define the class of polynomially verifiable languages $\mathcal{NP}$ by

$$\mathcal{VP} = \{V(A) \mid A \text{ is polynomial time verifier}\}.$$

Example: SAT Given boolean formula φ

? Is φ satisfiable ?

Theorem 1.9 (Nondeterminism ↔ certificate)

$\mathcal{NP} = \mathcal{VP}$.

Proof: $\mathcal{NP} \subseteq \mathcal{VP}$

$L \in \mathcal{NP} \rightarrow \exists$ nondet. TM M : $L(M) = L$

Construct verifier A

$(x, c) \in \Sigma^* \times \{0,1\}^*$

A interprets c as guide through the non-deterministic choices of M

M might choose one m paths

$\rightarrow \lceil \operatorname{ld}(m) \rceil$ bits from c tell us with path to take

$\rightarrow$ with given c, we simulate one possible computation of M.

M accepts x $\iff$ one computation of M accepts

$\Updownarrow$

one c guides right choices

$\Rightarrow V(A) = L$

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

given a verifier A : $V(A) = L$ polytime

construct a non-det. TM M

1. M generates non-deterministically a binary string c with $p(|x|)$ bits (p bound on $\text{time}_A(|x|)$)

2. simulate $A(x, c)$

3. copy output

$\Rightarrow L(M) = L$ $\square$

Glossary: Types of Problems

original $\mathcal{NP}$: languages = decision problems
$\mathcal{UP}$ yes/no

implicitly : certificate search problem

yes/no question, but in case of yes, also want proof

SAT : yes/no : is φ satisfiable?
if so, what is a satisfying assignment

Can can you solve the search problem whenever we can solve the decision prob.
easy: non-det TM can store certificate

but: unclear if we only allow black-box reuse of a decider

Definition 1.10 (Poly-time reductions)

L_2 is "more complex" than L_1

Let $L_1 \in \Sigma_1^\star$ and $L_2 \in \Sigma_2^\star$ be two languages.
We say L_1 *is polynomial reducible to* L_2 (notation $L_1 \leq_p L_2$), if there exists a (deterministic) algorithm A of polynomial time complexity that computes a mapping $\Sigma_1^\star \to \Sigma_2^\star$ such that

$$\forall x \in \Sigma_1^\star \ : \ x \in L_1 \Leftrightarrow A(x) \in L_2.$$

We call A the *polynomial time reduction of* L_1 *to* L_2. ◀

Definition 1.11 (NP-hard, NP-complete)

A language L is called $\mathcal{NP}$*-hard* if for all $U \in \mathcal{NP}$ we have $U \leq_p L$.
A language L is called $\mathcal{NP}$*-complete*, if additionally $L \in \mathcal{NP}$. ◀

Lemma 1.12 (All for one, and one for all)

If L is $\mathcal{NP}$-hard and $L \in \mathcal{P}$ we must have $\mathcal{P} = \mathcal{NP}$.

Proof:

L $\mathcal{NP}$-hard and $L \in \mathcal{P}$

$\rightsquigarrow$ $\exists$ det. poly-time TM M : $L(M) = L$

$U \in \mathcal{NP}$ $\rightsquigarrow$ $U \leq_p L$ $\rightsquigarrow$ $\exists$ det. poly-time alg. B : $x \in U \iff B(x) \in L$

suffices to show $\exists$ poly-time alg C that decides U

1. $B(x)$ (by simulation)
2. $M(B(x))$ (by simulation)

M accepts $\iff$ $B(x) \in L$ $\iff$ $x \in U$ in poly-time $\square$.

If L $\mathcal{NP}$-hard, it is probably intractable

does this exist?