Advanced Algorithmics

Strategies for Tackling Hard Problems

Sebastian Wild

Markus Nebel

Lecture 2

2017-04-24

Do $\mathcal{NP}$-complete problems exist at all? Yes!

Definition 1.13 (SAT)

For $\mathcal{F}$ the set of formulæ from propositional logic and $code : \mathcal{F} \longrightarrow \Sigma^\star$ a corresponding encoding over alphabet Σ the *satisfiability problem* (of propositional logic), SAT for short, is defined by following language:

$$\mathrm{SAT} \quad := \quad \{code(F) \in \Sigma^\star \mid F \text{ is a } \textbf{satisfiable} \text{ formula}\}.$$ ◀

Theorem 1.14 (Cook-Levin)

SAT is $\mathcal{NP}$-complete.

- SAT $\in \mathcal{NP} = \mathcal{VP}$ ✓ certificate = sat. assignment
- $\forall L \in \mathcal{NP}$ $L \leq_p$ SAT
  - → nondet. TM that runs poly-time $p(n)$
    - state at time t
    - symbol on tape at time t and pos i

forgot position of head on tape (oops)

Observation: $\leq_p$ is transitive, so SAT $\leq_p X \rightsquigarrow X$ is $\mathcal{NP}$-complete.

Further hard problems

Definition 1.15 (3SAT)

Given: formula ϕ in 3-CNF, i.e., $n, m \in \mathbb{N}$ and $l_{ij} \in \{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n\}$ for $i \in [m], j \in [3]$

Question: Is there a satisfying assignment $v : [n] \to \{0, 1\}$? ◀

Definition 1.16 (Vertex Cover)

Given: graph $G = (V, E)$ and $k \in \mathbb{N}$

Question: $\exists V' \subset V \quad : \quad |V'| \leq k \ \wedge \ \forall \{u, v\} \in E : \big(u \in V' \vee v \in V'\big)$ ◀

Definition 1.17 (Dominating Set)

Given: graph $G = (V, E)$ and $k \in \mathbb{N}$

Question: $\exists V' \subset V \quad : \quad |V'| \leq k \ \wedge \ \forall v \in E : \big(v \in V' \ \vee \ \exists u \in N(v) : u \in V'\big)$ ◀

Definition 1.18 (Hamiltonian Cycle)

Given: graph $G = (V, E)$ (directed and undirected version)

Question: Is there a vertex-simple cycle in G of length $|V|$? ◀

Further hard problems [2]

Definition 1.19 (Clique)

Given: graph $G = (V, E)$ and $k \in \mathbb{N}$

Question: $\exists V' \subset V \quad : \quad |V'| \geq k \ \wedge \ \forall u, v \in V' : \{u, v\} \in E$ ◀

Definition 1.20 (Independent Set)

Given: graph $G = (V, E)$ and $k \in \mathbb{N}$

Question: $\exists V' \subset V \quad : \quad |V'| \geq k \ \wedge \ \forall u, v \in V' : \{u, v\} \notin E$ ◀

Definition 1.21 (Traveling Salesperson (TSP))

Given: distance matrix $D \in \mathbb{N}^{n \times n}$ and $k \in \mathbb{N}$

Question: Is there a permutation $\pi : [n] \to [n]$ with $\sum_{i=1}^{n-1} D_{\pi(i),\pi(i+1)} + D_{\pi(n),\pi(1)} \ \leq \ k$? ◀

Definition 1.22 (Graph Coloring)

Given: graph $G = (V, E)$ and $k \in \mathbb{N}$

Question: $\exists c : V \to [k] \quad : \quad \forall \{u, v\} \in E : c(u) \neq c(v)$? ◀

Further hard problems [3]

Definition 1.23 (Set Cover)

Given: $n \in \mathbb{N}$, sets $S_1, \ldots, S_m \subseteq [n]$ and $k \in \mathbb{N}$

Question: $\exists I \subseteq [m] \quad : \quad \bigcup_{i \in I} S_i \;=\; [n] \;\wedge\; |I| \leq k$?

Definition 1.24 (Weighted Set Cover)

Given: $n \in \mathbb{N}$, sets $S_1, \ldots, S_m \subseteq [n]$, costs $c_1, \ldots, c_m \in \mathbb{N}_0$ and $k \in \mathbb{N}$

Question: $\exists I \subseteq [m] \quad : \quad \bigcup_{i \in I} S_i \;=\; [n] \;\wedge\; \sum_{i \in I} c_i \;\leq\; k$?

Definition 1.25 (Closest String)

Given: $s_1, \ldots, s_n \in \Sigma^m$ and $k \in \mathbb{N}$

Question: $\exists s \in \Sigma^m \quad : \quad \forall i \in [n] : d_H(s, s_i) \leq k$ $\qquad$ (d_H Hamming-distance)

Definition 1.26 (Max Cut)

Given: graph $G = (V, E)$ and $k \in \mathbb{N}$

Question: $\exists C \subset V \quad : \quad \big| E \cap \{\{u, v\} \mid u \in C, v \notin C\} \big| \geq k$

Further hard problems [4]

Definition 1.27 (Subset Sum)

Given: $x_1, \ldots, x_n \in \mathbb{Z}$

Question: $\exists I \subseteq [n] \ : \ \underline{I \neq \emptyset} \wedge \sum_{i \in I} x_i = 0$?

(missing in lecture)

Definition 1.28 ((0/1) Knapsack)

Given: $w_1, \ldots, w_n \in \mathbb{N}$, $v_1, \ldots, v_n \in \mathbb{N}$ and $b, k \in \mathbb{N}$

Question: $\exists I \subseteq [n] \quad : \quad \sum_{i \in I} w_i \leq b \ \wedge \ \sum_{i \in I} v_i \geq k$?

Definition 1.29 (Bin Packing)

Given: $w_1, \ldots, w_n \in \mathbb{N}$, $b \in \mathbb{N}$, $k \in \mathbb{N}$

Question: $\exists a : [n] \to [k] \quad : \quad \forall j \in [k] : \sum_{\substack{i=1,\ldots,n \\ a[i]=j}} w_i \ \leq \ b$?

Definition 1.30 (0/1 Integer Programming)

Given: integer linear program (ILP) $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$ and $c \in \mathbb{Z}^n$ and $k \in \mathbb{Z}$

Question: Is there $x \in \{0,1\}^n$ with $Ax \leq b$ and $c^T x \geq k$?

(erroneously was k in N in lecture)

1.1 Optimization Problems

Definition 1.31 (Optimization Problem)
An *optimization problem* is given by 7-tuple $U = (\Sigma_I, \Sigma_O, L, L_I, M, cost, goal)$ with

1. Σ_I an alphabet (called input alphabet),
2. Σ_O an alphabet (called output alphabet),
3. $L \subseteq \Sigma_I^\star$ the language of allowable problem instances (for which U is well-defined),
4. $L_I \subseteq L$ the language of actual problem instances for U (for those we want to determine U's complexity),
5. $M : L \to 2^{\Sigma_O^\star}$ and with $x \in L$, $M(x)$ is the set of all feasible solutions for x.
6. $cost$ is a cost function, which assigns for $x \in L$ each pair (u, x) with $u \in M(x)$ a positive real number,
7. $goal \in \{\min, \max\}$. ◀

Definition 1.32 (Optimal Solutions, Solution Algorithms)

Let $U = (\Sigma_I, \Sigma_O, L, L_I, M, cost, goal)$ an optimization problem. For each $x \in L_I$ a feasible solution $y \in M(x)$ is called *optimal for x and U*, if

$$cost(y, x) \quad = \quad goal\{cost(z, x) \mid z \in M(x)\}.$$

◀

An algorithm A is *consistent with U* if $A(x) \in M(x)$ for all $x \in L_I$.
We say *algorithm B solves U*, if

1. B is consistent with U and
2. for all $x \in L_I$, $B(x)$ is optimal for x and U. ◀

Examples

Natural examples: Problems above with an input parameter k.

Less immediate example:

Definition 1.33 (MAX-SAT)

Given: CNF-Formula $\phi = C_1 \wedge \cdots \wedge C_m$ over variables $x_1, \ldots, x_n$
Allowable (=Actual) Instances: encodings of ϕ
$M(\phi) = \{0,1\}^n$ (variable assignments)
$cost(u, x)$: # of satisfied clauses in u under given assignment x
$goal$ = max

$\max \ cost(u, x) = m$

Definition 1.34 (NPO)

$\mathcal{NPO}$ is the class if optimization problems $U = (\Sigma_I, \Sigma_O, L, L_I, M, cost, goal)$ with

1. $L_I \in \mathcal{P}$,
2. there is a polynomial p_U with
   a) $\forall x \in L_I \; \forall y \in M(x) : |y| \leq p_U(|x|)$ and
   b) there is a polynomial time algorithm which for all $y \in \Sigma_O^\star$, $x \in L_I$ with $|y| \leq p_U(|x|)$ decides whether $y \in M(x)$ holds, and
3. function *cost* can be computed in polynomial time.

↳ only integral cost functions

MAX-SAT $\in \mathcal{NPO}$

$|x| \geq n = |\text{solution}|$

Definition 1.35 (PO)

$\mathcal{PO}$ is the class of optimization problems $U = (\Sigma_I, \Sigma_O, L, L_I, M, cost, goal)$ with

1. $U \in \mathcal{NPO}$, and
2. there is an algorithm of polynomial time complexity which for all $x \in L_I$ computes an optimal solution for x and U. ◀

Glossary of Problem Types

	check value threshold	find value
single answer	decision problem	evaluation problem
solution / witness	search problem	optimization problem

costs

Complexities?

decision to search — often easy, unclear in general

Update: Indeed possible for any NP problem! (see Arora, Barak 2007)

threshold to evaluation: binary search ✓ costs $\in \mathbb{N}$

Definition 1.36 (Threshold Languages)

Let $U = (\Sigma_I, \Sigma_O, L, L_I, M, cost, goal)$ an optimization problem, $U \in \mathcal{NPO}$.
For $Opt_U(x)$ the cost of an optimal solutions for x and U we define the *threshold language for* U as

$$Lang_U \quad = \quad \begin{cases} \left\{(x,k) \in L_I \times \{0,1\}^\star \,\middle|\, Opt_U(x) \le k_2\right\}, & \text{if } goal = \min, \\ \left\{(x,k) \in L_I \times \{0,1\}^\star \,\middle|\, Opt_U(x) \ge k_2\right\}, & \text{if } goal = \max. \end{cases}$$

We say U *is $\mathcal{NP}$-hard*, if $Lang_U$ is $\mathcal{NP}$-hard.

Corollary 1.37 (Optimization is harder than Threshold)

Let U an optimization problem.
If $Lang_U$ is $\mathcal{NP}$-hard and if $\mathcal{P} \neq \mathcal{NP}$ holds, we have $U \notin \mathcal{PO}$.

Assume $U \in \mathcal{PO}$

→ $\exists$ poly-time algo A that computes optimal y of any instance $x \in U$

→ $Opt_U(x)$ can be computed in poly-time

→ $Opt_U(x) \leq$ threshold — " —
$\geq$

→ polytime algo $Lang_U$ $\underset{\text{NP-hard}}{\Rightarrow}$ $\mathcal{P} = \mathcal{NP}$ ↯

Lemma 1.38 (MAX-SAT)

MAX-SAT is $\mathcal{NP}$-hard.

3-CNF-SAT $\leq_p$ $Lang_{MAX\text{-}SAT}$

x is an encoding of ϕ in CNF, with m clauses

'compute' $(x, m) \in Lang_{MAX\text{-}SAT}$?

$\square$

Summary

- We have formalized the classic notion of intractable problems.
  - What is running time, what is “poly-time”?
  - Decision problems $\leftrightarrow$ (formal) languages
  - $\mathcal{P}$, $\mathcal{NP}$ via Turing machines $\leftrightarrow$ certificates and verifiers
  - For the typical case of optimization problems, there are different versions of the problem, but (in)tractability typically carries over.

$\rightsquigarrow$ We can mathematically prove a problem is intractable ($\mathcal{NP}$-hard).

… but how can we tackle hard problems anyway?

2 Pseudopolynomial Algorithms and Strong NP-hardness

Definition 2.1 (Integer-Input Problem)

A U for which we can encode any input as a sequence of integers is called an *integer-input problem*.

For any instance x of an integer-input problem, we write $\mathsf{MaxInt}(x)$ for the largest integer occurring in the input encoding. ◀

(As before, integers are encoded in binary.)

TSP, Knapsack, Bin Packing, ILP

Definition 2.2 (Pseudopolynomial algorithm)

Let U be an integer-input problem and A an algorithm that solves U.
A has *pseudopolynomial time for* U, if there is a polynomial p in two variables with

$$Time_A(x) \;=\; \mathcal{O}\Big(p\big(|x|, \mathrm{MaxInt}(x)\big)\Big),$$

for every instance x to U.

If $\mathrm{MaxInt}(x) \le h(|x|)$ polynomial h

$\leadsto$ poly-time!

Definition 2.3 (Value-Bounded Subproblem)

Let U be an integer-input problem and let $h : \mathbb{N} \to \mathbb{N}$ be weakly increasing.
The *h-bounded subproblem of U* (notation $\mathsf{Value}(h)_U$) is the problem which results from U by allowing only inputs x with $\mathsf{MaxInt}(x) \leq h(|x|)$. ◀

Theorem 2.4 (Pseudopolynomial is polynomial for small h)

Let U be an integer-input problem and A a pseudopolynomial algorithm for U.
Then for every polynomial h there is a polynomial algorithm for $\text{Value}(h)_U$.

Hence if U is a decision problem then $\text{Value}(h)_U \in \mathcal{P}$,
if U is an optimization problem then $\text{Value}(h)_U \in \mathcal{PO}$.

pseudopolynomial $\to$ A has time $O(p(|x|, MaxInt(x)))$

for $x \in \text{Value}(h)_U$ $\leadsto$ $MaxInt(x) \leq h(|x|) = O(|x|^c)$ $c \in \mathbb{N}$

$\Rightarrow$ $p(|x|, MaxInt(x)) = O(p(|x|, |x|^c)) = O(|x|^d)$ $d \in \mathbb{N}$

Definition 2.5 (Knapsack (Optimization Version))
Let a tuple $(w_1, \ldots, w_n, v_1, \ldots, v_n, b)$ of $2n+1$ positive integers be given, $n \in \mathbb{N}$.
We call b the *capacity* of the knapsack, w_i the *weight* and v_i the *profit* (value) of the i-th object, $1 \leq i \leq n$.
The *optimization problem* KNAPSACK asks to find a subset $T \subseteq \{1, 2, \ldots, n\}$ of items with maximal total cost $cost(T) = \sum_{i \in T} v_i$ such that T fits into the knapsack, i.e., $\sum_{i \in T} w_i \leq b$.

This presentation was unnecessarily cluttered; see end of file for improved version without storing T

Dynamic Programming Algorithm

minimal weight that you need

$A[(i, k)] = (w, T)$

profit to be reached exactly

items $1 \ldots i$

$$A[(i+1, k)] = \min \begin{cases} A[(i,k)], & \text{// don't take } i+1 \\ (w' + v_{i+1}, T' \cup \{i+1\}), & \text{if } A[(i, k - v_{i+1})] = (w', T') \neq \text{null and } w' + v_{i+1} \leq b \quad \text{// try take } i+1 \end{cases}$$

(update missing in lecture)

$$A[(1, k)] = \begin{cases} (0, \emptyset) & k = 0 \\ (w_1, \{1\}) & k = v_1 \\ \text{null} & \text{otherwise} \end{cases}$$

$A[(n,k)]$ check all entries with $k \leq \sum v_i = V$

Running time: # entries $n \times V$

time for each entry $O(1)$ (in uniform model; otherwise another log-factor)

$O(n \cdot V)$ $V \leq n \cdot MaxInt(x)$

$\Rightarrow$ DPKP has pseudopolynomial time

Theorem 2.6 (DP for Knapsack is pseudopolynomial)
For every instance I to KNAPSACK we have

$$Time_{DPKP}(I) \;=\; \mathcal{O}\big(|I|^2 \cdot \mathrm{MaxInt}(I)\big),$$ (log factor missing)

i.e., DPKP has pseudopolynomial time for KNAPSACK. ◀

Definition 2.7 (strongly NP-hard)

An integer-input problem is called *strongly $\mathcal{NP}$-hard*, if there exists a polynomial p such that $\mathsf{Value}(p)_U$ is $\mathcal{NP}$-hard.

So: strongly $\mathcal{NP}$-hard $\rightsquigarrow$ hard even for instances with small numbers.

Theorem 2.8 (strongly NP-hard → no pseudopoly. algorithm)
Let $\mathcal{P} \neq \mathcal{NP}$ and U a strongly $\mathcal{NP}$-hard (integer-input) problem.
Then there exists no algorithm with pseudopolynomial time for U. ◀

U strong $\mathcal{NP}$-hard $\rightarrow$ $\exists p$ $\mathrm{Value}(p)_U$ $\mathcal{NP}$-hard for polynomial p

If A was pseudopolynomial algo U

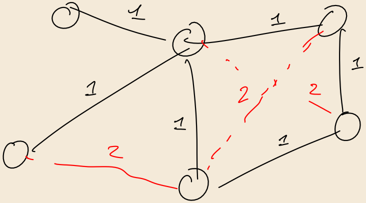
$\Rightarrow$ A runs in poly-time for $x \in \mathrm{Value}(q)_U$ (by Thm 2.4)
for any polynomial q

in part. for p $\overset{\rightarrow}{\mathrm{Value}(p)_U \ \mathcal{NP}\text{-hard}}$ $\mathcal{P} = \mathcal{NP}$ $\square$

A single polynomial p, so $\mathrm{Value}(p)_U$ is $\mathcal{NP}$-hard,
suffices to show $\mathrm{Value}(q)_U$ $\mathcal{NP}$-hard for any polynomial q.

Example: TSP is strong NP-hard

Hamilton $\le_p$ Value(2)$_{TSP}$



G has HC

$\Leftrightarrow$ tour of length n

Hamilton NP-hard

Improved Presentation of the DP algorithm for Knapsack

Definition 2.5 (Knapsack (Optimization Version))

Let a tuple $(w_1, \ldots, w_n, v_1, \ldots, v_n, b)$ of $2n+1$ positive integers be given, $n \in \mathbb{N}$.
We call b the *capacity* of the knapsack, w_i the *weight* and v_i the *profit* (value) of the i-th object, $1 \le i \le n$.
The *optimization problem* KNAPSACK asks to find a subset $T \subseteq \{1, 2, \ldots, n\}$ of items with maximal total cost $cost(T) = \sum_{i \in T} v_i$ such that T fits into the knapsack, i.e., $\sum_{i \in T} w_i \le b$.

Dynamic Programming

Key Idea: ith can be taken or not indep. of rest
BUT respect weight bound

→ fix the exact profit k for all $0 \le k \le V = \sum_{i=1}^{n} v_i$

$$A[i,k] = \min\{ W : \exists I \subseteq [i] : \sum_{i \in I} w_i = W \le b \wedge \sum_{i \in I} v_i = k \}$$

$$A[i,k] = \min \begin{cases} A[i-1, k] & \text{// don't pick } i \\ A[i-1, k-v_i] + w_i & \text{// pick } i \end{cases}$$

$$A[1,k] = \begin{cases} 0 & k = 0 \\ w_1 & k = v_1 \\ +\infty & \text{otherwise} \end{cases}$$

$A[n,k]$ start with $k = V$, go down until $A[n,k] \neq \infty$

optimal choice of items found by backtracing

Running time

\# entries $n \times V \leq n^2 \cdot \mathrm{MaxInt}(x)$

time to compute one entry $O(1)$ $O(\log(\mathrm{MaxInt}(x)))$

$V \leq n \cdot \mathrm{MaxInt}(x)$