



3

Pseudopolynomial Algorithms

13 May 2025

Prof. Dr. Sebastian Wild

Outline

3 Pseudopolynomial Algorithms

- 3.1 Integer Problems
- 3.2 Knapsack
- 3.3 Strong NP-hardness

3.1 Integer Problems

Integer Problems

Definition 3.1 (Integer-Input Problem)

A problem U for which (part of the) input is a *sequence of integers* is called an *integer-input problem*.

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

(As before, integers are encoded in binary.)

Examples

- ▶ TRAVELINGSALESMAN
- ▶ SUBSETSUM
- ▶ BINPACKING
- ▶ ILP
- ▶ KNAPSACK

Pseudopolynomial

Definition 3.2 (Pseudopolynomial algorithm)

\neq quasi-polynomial ($= 2^{O(\log^c n)}$)

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

$$\text{Time}_A(x) = O\left(p(|x|, \text{MaxInt}(x))\right),$$

for every instance x to U .

Note: If $\text{MaxInt}(x) \leq h(|x|)$ for a polynomial h , then
 $p(|x|, \text{MaxInt}(x)) \leq g(|x|)$ for a polynomial g .

Pseudopolynomial Languages

Definition 3.3 (Value-Bounded Subproblem)

Let U be an integer-input problem and let $h : \mathbb{N} \rightarrow \mathbb{N}$ be weakly increasing.

The *h -bounded subproblem of U* (notation $Value(h)_U$) is the problem which results from U by allowing only inputs x with $MaxInt(x) \leq h(|x|)$. ◀

Theorem 3.4 (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 polytime algorithm for $Value(h)_U$. ◀

Proof:

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

3.2 Knapsack

Knapsack (Optimization Version)

Definition 3.5 (Knapsack (Optimization Version))

Given: tuple $(w_1, \dots, w_n; v_1, \dots, v_n; b)$ of $2n + 1$ positive integers, $n \in \mathbb{N}$.

We call b the *capacity* of the knapsack, w_i the *weight* and v_i the *value* (profit) of the i -th object, $1 \leq i \leq n$.

Goal: The *optimization problem* *KNAPSACK* asks to find a subset $T \subseteq \{1, 2, \dots, n\}$ of items with maximal total value $cost(T) = \sum_{i \in T} v_i$ such that T fits into the knapsack, i. e., $\sum_{i \in T} w_i \leq b$.



Recap: The 6 Steps of Dynamic Programming

↪ see *Efficient Algorithms*

1. Define **subproblems** (and relate to original problem)
2. **Guess** (part of solution) ↪ local brute force
3. Set up **DP recurrence** (for quality of solution)
4. Recursive implementation with **Memoization**
5. Bottom-up **table filling** (topological sort of subproblem dependency graph)
6. **Backtracing** to reconstruct optimal solution

► Steps 1–3 require insight / creativity / intuition;
Steps 4–6 are mostly automatic / same each time

↪ Correctness proof usually at level of DP recurrence

👍 running time too! worst case time = #subproblems · time to find single best guess

Dynamic Programming Solution

- ▶ **Subproblems:** (n', b') : only items $1 \leq i \leq n'$ and total weight b'
- ▶ **Guess:** whether to include item n'
- ▶ **Recurrence:** $V[n', b'] = \max$ value in subproblem (n', b')

Pseudopolynomial Knapsack

Theorem 3.6 (DP for Knapsack is pseudopolynomial)

For every instance I to KNAPSACK we have

$$Time_{DPKP}(I) = O(|I| \cdot MaxInt(I) \log(MaxInt(I))),$$

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



Beyond Knapsack

- ▶ Similar trick works for some other **NP**-complete problems, e. g., PARTITION, MAKINGCHANGE
- ▶ for yet other **NP**-complete problems, e. g., TRAVELINGSALESMAN, no such algorithms seems to exist . . .

*. . . can we give evidence that likely **no** pseudopolynomial algorithm is **possible**?*

3.3 Strong NP-hardness

Hardness

Definition 3.7 (strongly NP-hard)

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

So: strongly NP-hard \rightsquigarrow hard even for instances with “small” numbers.

Theorem 3.8 (strongly NP-hard \rightarrow no pseudopoly. algorithm)

Let $P \neq NP$ and U a strongly NP-hard (integer-input) problem.

Then there exists no algorithm with pseudopolynomial time for U . ◀

Proof:

Example

Theorem 3.9

TRAVELINGSALESMAN is strongly NP-hard.

Proof:



It's all about the encoding

Theorem 3.10 (strongly hard iff unary hard)

An integer-input problem is strongly NP-hard if, and only if, representing its instances with unary encoding for integers remains NP-hard. ◀

Proof:

A strongly NP-hard $\rightsquigarrow \exists$ polynomial p s.t. $Value(p)_A$ NP-hard

For $x \in Value(p)_A$, unary encoding blows up $|x|$ only by polynomial factor.

$\rightsquigarrow A$ encoded with unary numbers NP-hard.

Conversely, let A with unary numbers be NP-hard.

With unary encoding, $MaxInt(x) \leq |x|$, so $Value(n \mapsto n)_A = A$ is NP-hard. ■

Summary

- 👍 Pseudopolynomial algorithms can be practically efficient if numbers are (really) small
- 👎 Only applicable to few problems