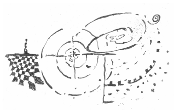
# Algorithmic Adventures

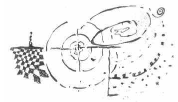
## From Knowledge to Magic



Book by Juraj Hromkovič, ETH Zurich
Slides by Tom Verhoeff, TU Eindhoven



## Quotation



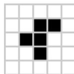
There is no greater loss than time which has been wasted

Michelangelo Buonarroti

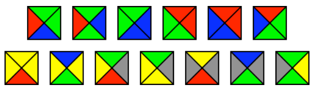


## Undecidability Is Not Rare

- Decide* whether a Game of Life configuration stabilizes



- Decide whether a set of Wang tiles can tile the plane



- Decide whether a Diophantine equation (multivariable polynomial equation, like $a^3 + b^3 = c^3$) has a solution in integers
- Decide whether a program has a specific non-trivial property, like whether it always halts, always outputs 0, ... [cf. Rice's Theorem]

*In each case, the algorithm needs to work for *all* possible inputs (shown in yellow). All these decision problems turn out to involve a *universal* mechanism.

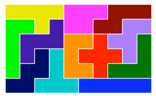


## Some Algorithms Are Very Inefficient

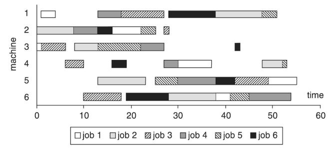
For some *algorithmically solvable* problems, our algorithmic solutions turn out to be very slow

Slow algorithms are *practically* unusable:

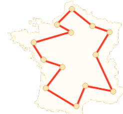
- Packing puzzles



- Scheduling jobs on machines




- Traveling Salesman Problem (**TSP**): find shortest tour visiting each town in a given set, given their distances



How can we investigate this phenomenon?

How can we overcome this limitation?

## Algorithmic Complexity

The **time complexity** of algorithm $A$ on input $I$:

number of instructions performed in computation of $A$ on $I$

The **space complexity** of algorithm $A$ on input $I$:

amount of memory used in computation of $A$ on $I$

Complexity varies with **size of the input** (amount of input data)

The **time complexity** of algorithm $A$ **as function of input size**:

$\mathit{Time}_A(n)$ = *worst-case* number of instructions performed in computation of $A$ on any input of size $n$



## Asymptotic Algorithmic Time Complexity

The function $\mathit{Time}_A(n)$ also depends on details of the programming language and implementation of the algorithm as program

**Definition** Function $f(n) \geq 0$ is $\mathcal{O}(g(n))$ ('$f$ is big oh of $g$') when

$f(n) \leq C \cdot g(n)$ for some constant $C$ and all sufficiently large $n$

Example: $10n^2 + 7n + 20$ is $\mathcal{O}(n^2)$, but not $\mathcal{O}(n)$ and not $\mathcal{O}(\log n)$

The **asymptotic time complexity** of algorithm $A$ is $f(n)$:

$\mathit{Time}_A(n)$ is $\mathcal{O}(f(n))$ and $f(n)$ is $\mathcal{O}(\mathit{Time}_A(n))$

The asymptotic complexity is *robust*, independent of implementation

**Complexity classes**: Constant, Logarithmic, Linear, Linearithmic $\mathcal{O}(n \cdot \log n)$, Quadratic, Cubic, ..., Polynomial, Exponential, ...



## Asymptotic Time Complexity Examples

| Complexity | Name | Example* |
|---|---|---|
| $\mathcal{O}(1)$ | Constant | Determine whether $n$-bit number is even |
| $\mathcal{O}(\log n)$ | Logarithmic | Find item in sorted list by *Binary Search* |
| $\mathcal{O}(n)$ | Linear | Find item in list by *Linear Search* |
| $\mathcal{O}(n \log n)$ | Linearithmic | Sort list by *Merge Sort* |
| $\mathcal{O}(n^2)$ | Quadratic | Sort list by *Bubble Sort* |
| $\mathcal{O}(n^k)$ | Polynomial | Determine whether $n$-bit number is *prime* |
| $\mathcal{O}(2^n)$ | Exponential | Solve TSP by *Dynamic Programming* |
| $\mathcal{O}(n!)$ | Factorial | Solve TSP by *Brute Force Search* |

*The input is a list of $n$ elements (possibly bits)



## What Is the Limit of Practical Solvability?

| $n$ / $f(n)$ | 10 | 50 | 100 | 300 |
|---|---|---|---|---|
| $10n$ | 100 | 500 | 1000 | 3000 |
| $2n^2$ | 200 | 5 000 | 20 000 | 180 000 |
| $n^3$ | 1000 | 125 000 | 1 000 000 | 27 000 000 |
| $2^n$ | 1024 | 16 digits | 31 digits | 91 digits |
| $n!$ | $\approx 3.6 \cdot 10^6$ | 65 digits | 158 digits | 615 digits |

A problem is called **tractable** when it can be solved by a **polynomial** algorithm (asymptotic time complexity is $\mathcal{O}(n^k)$ for some constant $k$)

**P** denotes the class of all **polynomial decisions problems**

## How Much More Can You Do on a $2\times$ Faster Machine?

Assume $n = 100$ takes 1 hour on machine $A$.
How much further do you get on a $2\times$ faster machine $B$ in 1 hour?

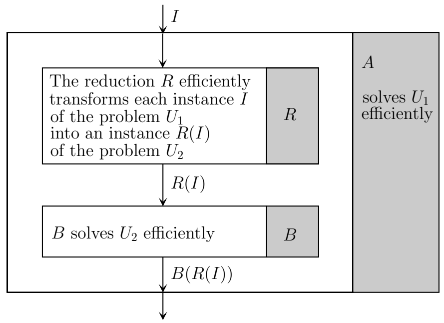
| | Time | $n$ on $A$ | $n$ on $B$ | More on $B$ | Factor |
|---|---|---|---|---|---|
| Logaritmic | $C_1 \log_2 n$ | 100 | 10000 | 9900 | 100 |
| Linear | $C_2\, n$ | 100 | 200 | 100 | 2 |
| Linearitmic | $C_3\, n \log_2 n$ | 100 | 178 | 78 | 1.78 |
| Quadratic | $C_4\, n^2$ | 100 | 141 | 41 | 1.41 |
| Cubic | $C_5\, n^3$ | 100 | 126 | 26 | 1.26 |
| Exponential | $C_6\, 2^n$ | 100 | 101 | 1 | 1.01 |



## Polynomial-time Reduction

Algorithm $R$ is a **polynomial reduction** from problem $U_1$ to $U_2$ when

- $R$ is a *polynomial* algorithm, and
- the solution for instance $I$ of problem $U_1$ equals the solution for instance $R(I)$ of problem $U_2$, for all instances $I$ of $U_1$






## Polynomial-time Reduction

By definition, the following statements are equivalent:

- Problem $U_1$ is **polynomial-time reducible** to problem $U_2$
- There exists a polynomial reduction $R$ from $U_1$ to $U_2$
- $U_1 \leq_{\text{pol}} U_2$
- Problem $U_1$ is *polynomially no harder than* problem $U_2$

An example follows



## Knapsack Problem

**Subset Sum Problem**, or (simplified) **Knapsack Problem**:

> For a given positive integer $K$ and set $S$ of items $x$ with positive integer size $s(x)$, does there exists a subset $T$ of $S$ whose total size $\sum_{x \in T} s(x)$ equals $K$?

$K$ is the size of the knapsack, $S$ contains the items to pack, and $s$ gives their sizes.

The question is whether the knapsack can be filled exactly with a suitable selection $T$ of the items.

Example: item sizes 110, 90, 70, 50, 30, 30, 20, and $K = 150$

## Settling Debts Problems

A group of friends lend each other money throughout the year. They carefully record each transaction. When Alice lends 10 euro to Bob, this is recorded as Alice $\xrightarrow{10}$ Bob.

At the end of the year they wish to settle all their debts. Money can be transferred between any *pair* of persons.

Problem variants:

- minimize the number of transfers
- minimize the total amount transferred
- minimize both



## Reduce Knapsack to Settling Debts

Given an instance $I$ for Knapsack, construct an instance $R(I)$ for Settling Debts: $|S|$ positive balances $s(x)$ for $x \in S$, and two negative balances $-K$ and $K - \sum_{x \in S} s(x)$. N.B. The total balance = 0.

| | | | | | | |
|---|---|---|---|---|---|---|
| +110 | +90 | +70 | +50 | +30 | +30 | +20 |
| | −150 | | | −250 | | |

The instance $R(I)$ requires at least $|S|$ transfers to settle, since each positive balance needs an outgoing transfer. A settling of all debts for $R(I)$ with $|S|$ transfers exists if and only if there exists a subset $T$ of $S$ whose total size equals $K$, that is, when it solves $I$.

Thus: Knapsack $\leq_{\text{pol}}$ Settling Debts *in minimum number of transfers*



## Using Polynomial-time Reducibility $U_1 \leq_{\text{pol}} U_2$

(Compare to *algorithmic* reducibility and its uses, in Ch. 4)

If we know $U_1 \leq_{\text{pol}} U_2$, then this can be used in two ways:

1. Polynomial solvability of $U_2$ implies polynomial solvability of $U_1$ (Note the order of $U_2$ and $U_1$ here)
2. If $U_1$ can*not* be solved by a polynomial algorithm, then $U_2$ can*not* be solved by a polynomial algorithm

Many problems for which we have not found polynomial algorithms are polynomially equally hard: $U_1 \leq_{\text{pol}} U_2$ and $U_2 \leq_{\text{pol}} U_1$

These problems are called **NP-hard**

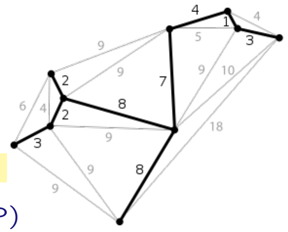
Knapsack (Subset Sum) is known to be NP-hard
Hence, Settling Debts *in minimum number of transfers* is NP-hard



## Easy/Hard Pairs

- *Hard*: Determine whether a graph has a **Hamiltonian circuit** that visits each **vertex** exactly once

  *Easy*: Determine whether a graph has an **Euler circuit** that visits each **edge** exactly once




- *Hard*: Determine a settling of all debts, that minimizes the number of transfers

  *Easy*: Determine a settling of all debts, that minimizes the total amount transferred

- *Hard*: **Traveling Salesman Problem** (TSP)

  *Easy*: Determine a **Minimum Spanning Tree** (MST) of a connected, edge-weighted graph: a set of edges of minimum total weight that connects all vertices (this is a *tree*; see figure)

## Settling Debts, Minimizing Total Amount Transferred, Is Easy

Here is a greedy* algorithm:

1. Determine the balance $b_i$ for each person
2. While there is still someone with a nonzero balance, do:
   (a) Select any person $i$ with $b_i < 0$, and any person $j$ with $b_j > 0$
   (b) Let $m$ be the minimum of $-b_i$ and $b_j$; hence, $m > 0$
   (c) Include transfer $i \xrightarrow{m} j$ in the settlement
   (d) Increase $b_i$ by $m$ and decrease $b_j$ by $m$
3. All $b_k = 0$, hence the included transfers settle all debts

*Step 2a makes it greedy: settle maximally among the first candidate pair found



## Settling Debts, Minimizing Total Amount Transferred: Proof

$\sum_k b_k = 0$ holds initially and after every iteration of Step 2.

Step 2a is always possible, because $\sum_k b_k = 0$ and not all $b_k = 0$.

The repetition of Step 2 terminates, because in each iteration at least one nonzero $b_k$ is reduced to zero by Step 2d.

Therefore, the number of transfers is at most $N$ (number of persons). In fact, it is at most $N - 1$, because the final *two* nonzero balances cancel each other in a *single* transfer.

Let $P$ be the total amount of the positive balances, and $N$ the total amount of the negative balances. Hence, $P = -N$. The *minimum total amount to be transferred* equals $P$.

The total amount transferred equals $P$, and hence is minimal.



## Summary

- **Time complexity** of an algorithm: how many steps it takes to compute an answer, in relation to input *size* (worst-case)
- **Space complexity** of an algorithm: how many variables it takes to compute an answer, in relation to input *size* (worst-case)
- **Complexity classes** defined in terms of *asymptotic* complexity: Polynomial time (**P**), Exponential time (**EXP**), . . .
- **NP decision problem** ≈ YES answer *verifiable* in polynomial time
- **NP-hard**: class of *hardest* NP problems (**polynomial reduction**)
- **P $\stackrel{?}{=}$ NP**: Can all NP problems be solved in polynomial time?
- Today we know only *exponential* algorithms for NP-hard problems: **intractable**, practically unsolvable for larger inputs; not hopeless