Chapter 5: Hard Problems

Algorithmic Adventures
From Knowledge to Magic

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

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 overcome this limitation?

Quotation

There is no greater loss than time which has been wasted
Michelangelo Buonarroti
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:

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

Asymptotic Algorithmic Time Complexity

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

**Definition** Function $f(n) \geq 0$ is $O(g(n))$ (‘$f$ 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 $O(n^2)$, but not $O(n)$ and not $O(\log n)$

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

- $\text{Time}_A(n)$ is $O(f(n))$ and $f(n)$ is $O(\text{Time}_A(n))$
- The asymptotic complexity is robust, independent of implementation

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

Asymptotic Algorithmic Time Complexity Examples

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Name</th>
<th>Example*</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(1)$</td>
<td>Constant</td>
<td>Determine whether $n$-bit number is even</td>
</tr>
<tr>
<td>$O(\log n)$</td>
<td>Logarithmic</td>
<td>Find item in sorted list by Binary Search</td>
</tr>
<tr>
<td>$O(n)$</td>
<td>Linear</td>
<td>Find item in list by Linear Search</td>
</tr>
<tr>
<td>$O(n \log n)$</td>
<td>Linearithmic</td>
<td>Sort list by Merge Sort</td>
</tr>
<tr>
<td>$O(n^2)$</td>
<td>Quadratic</td>
<td>Sort list by Bubble Sort</td>
</tr>
<tr>
<td>$O(n^k)$</td>
<td>Polynomial</td>
<td>Determine whether $n$-bit number is prime</td>
</tr>
<tr>
<td>$O(2^n)$</td>
<td>Exponential</td>
<td>Solve TSP by Dynamic Programming</td>
</tr>
<tr>
<td>$O(n!)$</td>
<td>Factorial</td>
<td>Solve TSP by Brute Force Search</td>
</tr>
</tbody>
</table>

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

What Is the Limit of Practical Solvability?

<table>
<thead>
<tr>
<th>$n$</th>
<th>10</th>
<th>50</th>
<th>100</th>
<th>300</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(n)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$10n$</td>
<td>100</td>
<td>500</td>
<td>1000</td>
<td>3000</td>
</tr>
<tr>
<td>$2n^2$</td>
<td>200</td>
<td>5000</td>
<td>20000</td>
<td>180000</td>
</tr>
<tr>
<td>$n^3$</td>
<td>1000</td>
<td>125000</td>
<td>1000000</td>
<td>27000000</td>
</tr>
<tr>
<td>$2^n$</td>
<td>1024</td>
<td>16 digits</td>
<td>31 digits</td>
<td>91 digits</td>
</tr>
<tr>
<td>$n!$</td>
<td>$\approx 3.6 \cdot 10^6$</td>
<td>65 digits</td>
<td>158 digits</td>
<td>615 digits</td>
</tr>
</tbody>
</table>

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

**$P$** denotes the class of all polynomial decisions problems
How Much More Can You Do on a 2x Faster Machine?

Assume \( n = 100 \) takes 1 hour on machine \( A \).

How much further do you get on a 2x faster machine \( B \) in 1 hour?

<table>
<thead>
<tr>
<th>Time</th>
<th>( n \text{ on } A )</th>
<th>( n \text{ on } B )</th>
<th>More on ( B )</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logarithmic</td>
<td>( C_1 \log_2 n )</td>
<td>100</td>
<td>9900</td>
<td>100</td>
</tr>
<tr>
<td>Linear</td>
<td>( C_2 n )</td>
<td>100</td>
<td>100</td>
<td>2</td>
</tr>
<tr>
<td>Linear ( n \log_2 n )</td>
<td>( C_3 n \log_2 n )</td>
<td>100</td>
<td>78</td>
<td>1.78</td>
</tr>
<tr>
<td>Quadratic</td>
<td>( C_4 n^2 )</td>
<td>100</td>
<td>41</td>
<td>1.41</td>
</tr>
<tr>
<td>Cubic</td>
<td>( C_5 n^3 )</td>
<td>100</td>
<td>26</td>
<td>1.26</td>
</tr>
<tr>
<td>Exponential</td>
<td>( C_6 2^n )</td>
<td>100</td>
<td>1</td>
<td>1.01</td>
</tr>
</tbody>
</table>

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 \( U_1 \).

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 Bob 10 euros, 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 $= 0$.

The instance $R(I)$ requires at least $|S|$ transfers to settle, since each positive balance needs an outgoing transfer. A settlement 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 $\leq_{pol}$ Settling Debts in minimum number of transfers

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

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

If we know $U_1 \leq_{pol} U_2$, 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 $U_1$ cannot be solved by a polynomial algorithm, then $U_2$ cannot be solved by a polynomial algorithm

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

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 $i \rightarrow j$ 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 ($\mathcal{P}$), Exponential time ($\mathcal{EXP}$), …
- **NP decision problem**: $\approx$ YES answer verifiable in polynomial time
- **NP-hard**: class of hardest NP problems (polynomial reduction)
- **$\mathcal{P} \neq \mathcal{NP}$**: 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