Introduction to

Linear and Combinatorial Optimization

9 Efficient Algorithms

9.1 Basic Definitions

Efficient Algorithms

What is an efficient algorithm?

- efficient: running time bounded by a polynomial of the input size
- algorithm: Turing Machine or other formal model of computation

Simplified Definition

An algorithm consists of

- "elementary steps" like, e.g., variable assignments
- simple arithmetic operations

which only take a constant amount of time. The running time of the algorithm on a given input is the number of such steps and operations.

Bit Model and Arithmetic Model

Two ways of measuring the running time and the size of the input I of A:

Bit Model

- count bit operations
    - e.g., adding two n-bit numbers takes $n + 1$ steps
    - e.g., multiplying two n-bit numbers takes $O(n^2)$ steps
- size of input I is the total number of bits needed to encode "structure" and numbers

Arithmetic Model

- simple arithmetic operations on arbitrary numbers can be performed in constant time
    - e.g., adding two numbers takes 1 step
    - e.g., multiplying two numbers takes 1 step
- size of input I is total number of bits needed to encode "structure" plus # numbers in the input

Polynomial vs. Strongly Polynomial Running Time

Definition 9.1

i An algorithm runs in polynomial time if, in the bit model, its (worst-case) running time is polynomially bounded in the input size.

ii An algorithm runs in strongly polynomial time if, in the bit model as well as in the arithmetic model, its (worst-case) running time is polynomially bounded in the input size.

Examples:

- Prim's and Kruskal's Algorithm as well as the Ford-Bellman Algorithm and Dijkstra's Algorithm run in strongly polynomial time
- Euclidean Algorithm runs in polynomial time but not in strongly polynomial time:
  - $\gcd(a, b) = \begin{cases} a & \text{if } b = 0 \\ \gcd(b, a \bmod b) & \text{otherwise} \end{cases}$ for $a \geq b$
  - after two iterations $\gcd(a, b) = \gcd(\underbrace{a \bmod b}_{<a/2}, \underbrace{b \bmod (a \bmod b)}_{<b/2})$
  - $O(\log a)$ iterations suffice

Example: Arithmetic Model vs. Bit Model

Consider the following algorithm:

Given: n numbers $a_1, \ldots, a_n \in \mathbb{Z}_{>0}$

```
for i = 1 to n:
    a_1 := a_1 · a_1
output a_1
```

Arithmetic Model:

- Input size is n; running time is $O(n)$ (polynomial, even linear).

Bit Model:

- Input size is $\sum_{i=1}^{n}(\lfloor \log a_i \rfloor + 1)$
- Encoding size of the computed output ${a_1}^{2^n}$ is $\lfloor 2^n \log a_1 \rfloor + 1$.
- Output size can thus be exponential in the input size (e.g., if $a_1 \geq a_i$ for all $i = 1, \ldots, n$).
- Notice that the output size is a lower bound on the running time.

Pseudopolynomial Running Time

- In the bit model, we assume that numbers are binary encoded, i.e., the encoding of the number $n \in \mathbb{N}$ needs $\lfloor \log n \rfloor + 1$ bits.
- Thus, the running time bound $O(C\,n^2)$ of Ford's Algorithm where $C := 2\max_{a \in A} |c_a| + 1$ is not polynomial in the input size.
- If we assume, however, that numbers are unary encoded, then $C\,n^2$ is polynomially bounded in the input size.

Definition 9.2 An algorithm runs in pseudopolynomial time if, in the bit model with unary encoding of numbers, its (worst-case) running time is polynomially bounded in the input size.

Example:

Checking whether a given number $a \in \mathbb{Z}_{\geq 2}$ is prime by testing for all $1 < b < a$ whether b divides a is a pseudopolynomial time algorithm.