Types of Complexity Measures of Algorithms

- *Static* complexity measures depend only on an algorithm that is measured.

There are two *dynamic* complexity measures

- *Functional* complexity measures depend on an algorithm that is measured as well as on the input and output.
- *Processual* complexity measures depend on an algorithm, its realization, and on the input.

Example 1. The least number of the lines in the description of an algorithm is a static complexity measure.

Example 2. The least length of the description of an algorithm is a static complexity measure.

Static complexity measures are constants, that is, for each algorithm *A*, its static complexity measure is a number.

Example 3. The maximal size of the output of an algorithm when its input has the size *n* is a functional complexity measure.

Example 4. The maximal size of the output plus the size of the input of an algorithm when its input has the size *n* is a functional complexity measure.

Functional complexity measures are functions, that is, for each algorithm *A*, its functional complexity measure is a function.

Thus, to know which algorithms are better with respect to functional complexity, it is necessary to compare functions.

Example 5. An important type of processual complexity measures is **Computational Complexity** of algorithms, which measures resources utilized by the algorithm.

Examples:

Time complexity $T_A(x)$ = time that algorithm A needs to solve the problem with the input x

Space complexity $S_A(x)$ = memory that algorithm A needs to solve the problem with the input x

Processual complexity measures are functions, that is, for each algorithm A, its functional complexity measure is a function.

Thus, to know which algorithm is better with respect to processual complexity, for example, time complexity, it is necessary to compare functions.

Group complexity

Worst-case complexity

$$T_{Aw}(n) = \max \{T_A(x);\ l(x) = n\}$$

Average complexity

$$T_{Aa}(n) = \text{average} \{T_A(x);\ l(x) = n\}$$

Best-case complexity

$$T_{Ab}(n) = \min \{T_A(x);\ l(x) = n\}$$

Theorem. For any strictly increasing recursive function f, there is a recursive function g taking values in the set {1, 0} such that any computation of g by a Turing machine has time complexity larger than f, that is, $T_T(x) > f(x)$ for all Turing machines T and all x.

All these measures are called **direct complexity measures** of algorithms.

There are also **dual complexity measures**. They measure complexity of the results of algorithms as well as of the problems solved by algorithms.

The most popular dual complexity measure is called *algorithmic complexity* or *Kolmogorov complexity*.

Informally, it is defined as the length of the shortest program, which is necessary compute the given result.

Traditionally the theory of Kolmogorov complexity was developed top down: from larger classes to smaller classes of algorithms that were more relevant to computational problems. At first, as the history tells us, Kolmogorov complexity $C(x)$ was defined and studied independently for the class of all recursive algorithms by three mathematicians/computer scientists: Solomonoff (1964), Kolmogorov (1965), and Chaitin (1966).

Definition 1. The *Kolmogorov*(*algorithmic*) *complexity* $C(x)$ of an object (word) x is defined as

$$C(x) = \min \{l(p);\ U(p) = x\}$$

where $l(p)$ is the length of the word p and U is a universal algorithm in the class **R**.

This measure is called *absolute* Kolmogorov complexity because Kolmogorov complexity has one more form, which called relative.

Definition 2. The *relative* to a given word *y Kolmogorov complexity* $C(x \mid y)$ of an object (word) *x* is defined as

$$C(x \mid y) = \min \{l(p);\ U(p, y) = x\}$$

where $l(p)$ is the length of the word *p* and *U* is a universal algorithm in the class **R**.

This measure is called *relative* Kolmogorov complexity.

Asymptotically optimal function $f(x)$ in a set **F** of functions:

$$\exists\, k\ \forall g \in \mathbf{F}\ \forall\, x\ (f(x) \leq g(x) + k)$$

Theorem (Kolmogorov–Chaitin)**.** For all algorithmic complexities on a universal class of recursive algorithms, there is an optimal algorithmic complexity.

Conventional approach interprets $C(x)$ as the quantity of information in *x* and $C(x \mid y)$ as the quantity of information in *y* about *x*.

Correct interpretation: $C(x)$ as the quantity of information for building (computing) *x* and $C(x \mid y)$ as the quantity of information in *y* for building (computing) *x*.

The Library Metaphor

Types of problems:

1. Undecidable/unsolvable
2. Solvable/decidable
3. Tractable

A problem is **solvable** if there is an algorithm that can solve it.
A problem is **tractable** if there is an algorithm that has admissible complexity and can solve it.
Usually, it's mostly time tractability.
A problem is **tractable** if it is actually possible to find solutions to such a problem in a reasonable amount of time.

$T_A(n) = O(p(n))$
Problems with the deterministic polynomial time complexity form the class **P**.

Problems with the nondeterministic polynomial time complexity form the class **NP**.

P =NP ?