Computational complexity

• Motivation
• Some hard problems: Hamiltonian cycle, set partition, satisfiability of formulas
• Deterministic- and nondeterministic-time problem classes
• Polynomial time complexity
• Intractable (NP-hard) problems

Motivation

• Recall from Data Structures that complexity of a problem is the time function of the fastest algorithm that solves the problem
• Example: No search algorithm is better than \( O(n) \) for arbitrary collections, so the complexity of search is \( O(n) \)
• Some problems are decidable but take too long to solve in practice
• In this topic we characterize those intractable problems mathematically
Hamiltonian-cycle problem

- **Problem:** Given a graph $G = \langle V, E \rangle$, is there a path in $E^*$ that starts and ends with the same vertex and that passes through every vertex exactly once?

- **One solution** is to generate and test every possible sequence of $|V| + 1$ vertices
  - What is the complexity of the test operation?
  - What is the complexity of generate-and-test?
  - Can you think of a better algorithm?

Two more combinatorial problems

1. **Set partition:** Given a set $S \subseteq \mathbb{N}$, can $S$ be partitioned into two sets, $A$ and $B$, s.t. $\Sigma_{x \in A} x = \Sigma_{y \in B} y$?

2. **Satisfiability (SAT):** Given a formula $\phi$ in propositional logic (logic with $\neg, \land, \lor, \Rightarrow$, no predicates, no quantifiers), does a set of variable assignments exist that satisfies $\phi$ (makes $\phi$ true)?
   
   Examples: (a) $p \land q \land r$ (b) $\neg(p \land q \lor \neg q)$
Observations about these problems

- Each seems to require generation and checking of a very large set of candidate solutions
- Checking a given candidate solution is easy enough
- But the number of candidates seems to be \( O(2^n) \)
- For \( n > 100 \), the time required would be too much for practical purposes

Deterministic time complexity

- \[ DTIME(T(n)) = \{ L \subseteq \Sigma^* | L \text{ is accepted by a deterministic Turing machine in time } T(n) \} \]
- \( DTIME \) is a complexity class of problems
- \( SAT \) seems to be in \( DTIME(2^n) \) because \( O(2^n) \) candidate solutions seem to be required
Nondeterministic time complexity

- \( NTIME(T(n)) = \) 
  \( \{ L \in \Sigma^* \mid L \text{ is accepted by a } \text{nondeterministic Turing machine in time } T(n) \} \)
- To solve \( SAT \), a nondeterministic machine could “luckily” start by testing a set of variable assignements that satisfy the formula and immediately answer “yes”
- \( SAT \) is in \( NTIME(n) \) for propositional formulas of length \( n \), because evaluation of formulas is linear-time

Polynomial time complexity

- \( \mathcal{P} = \bigcup_{k \in \mathbb{N}} DTIME(n^k) \)
- That is, \( \mathcal{P} \) is the set of problems decidable in \( O(n^k) \) time (polynomial time), where \( n \) is the size of the problem and \( k \) is a constant
- \( \text{Examples:} \)
  - Searching a collection, \( O(n) \)
  - Sorting an array, \( O(n \lg n) \)
  - Searching a BST, \( O(\lg n) \)
  - Generating a graph reachability matrix, \( O(n^3) \)
The $NP$ complexity class

- $NP = \bigcup_{k \in \mathbb{N}} \mathrm{NTIME}(n^k)$
- That is, $NP$ is the set of problems decidable in $O(n^k)$ time on a nondeterministic TM
- *Intuition:* $NP$ problems are those for which a particular solution, once found, can be verified in polynomial time

$NP$ completeness

- For many problems in NP, e.g., SAT, no polynomial time solution is known
- Problems to which SAT or similar problems are reducible are called $NP$-complete
- If any of these has a polynomial-time solution, then all to, hence $P = NP$
- Most researchers think $P \neq NP$, i.e., NP-complete problems are believed to have no polynomial-time solutions
Intractability

- Certain problems are thought or known to have only exponential-time, $O(2^n)$, solutions
- $EXPTIME$ and $NPC$ are the $NP$-hard problems, considered intractable
- Problems of planning, scheduling, routing, drawing inferences, understanding language, etc., are in general intractable
- What to do: Replace intractable problem with a simpler one, e.g., one with a probabilistic or approximate solution

Sources