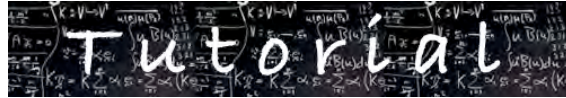




Christof Teuscher

The Complete Idiot's



to *In Theorio*, *In Silico*, and *In Vivo* Computation

Los Alamos National Laboratory
Advanced Computing Laboratory (CCS-1)

www.teuscher.ch/christof
christof@teuscher.ch



LA - UR - 06 - 3755
UNCLASSIFIED



Christof Teuscher • christof@teuscher.ch



Part IV

Complexity theory



LA - UR - 06 - 3755
UNCLASSIFIED





Readings

- M. Sipser. **Introduction to the Theory of Computation**. Thomson, 2nd ed., 2006
- J. Hromkovic. **Theoretical Computer Science**. Springer, 2004.
- C. H. Papadimitriou. **Computational Complexity**. Addison Wesley, 1994.



Complexity Theory

- Even if a problem is computationally solvable in principle, it may not be solvable in practice if the solution requires an inordinate amount of time or resources.



- **Complexity theory** deals with how hard/easy it is to solve computational problems.
- **Time** versus **space** complexity.



Measuring Complexity

- Example: $L = \{0^k1^k \mid k \geq 0\}$ is a decidable language, but how much time does it take a single-tape TM M to decide L ?
- We can count the number of steps M uses.
 - Worst-case?
 - Minimum?
 - Average?
- This can be very complex!
- The measure should be independent of the implementation.



The Big-O Notation

- **Asymptotic analysis** seeks to understand the running time for large inputs.
- Considers only the higher order terms because they dominate on larger inputs.
- Example:
 - Running time is given by $f(n) = 6n^3 + 2n^2 + 20n + 45$, where n is the input size.
 - We say that f is asymptotically at most n^3
 - n^3 is an **asymptotic upper bound**
 - $f(n) = O(n^3)$
- Intuitively: $f(n) = O(g(n))$ means that f is less or equal to g if we ignore differences up to a constant factor.



Class P (Polynomial)

- P is the class of languages that are **decidable in polynomial** time on a deterministic single-tape Turing machine.
- In other words: $P = \bigcup_k \text{TIME}(n^k)$
- Calling polynomial time the threshold of practical solvability has proven to be useful.
- Every context-free language is a member of P.
- Example: the *PATH* problem consists in determining whether there is a directed path from node *s* to *t* in a directed graph. $\text{PATH} \in P$. See polynomial time algorithm in Sipser, pp. 260.



Class NP (Non-deterministic Polynomial)

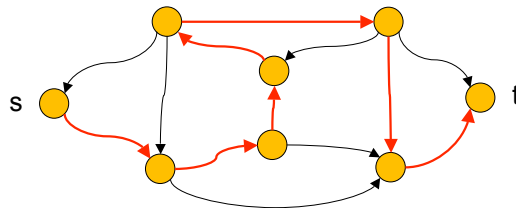
- Some problems do not seem to have any polynomial time algorithm. Why? The answer is open (P=NP?!)
 - A P time algorithm might yet be discovered, or
 - it might simply be that they *cannot* be solved in polynomial time.
- **NP** is the class of decision problems
 - that is solvable in polynomial time on a non-deterministic TM;
 - that are verifiable by a DTM in P time.
- P is a subset of NP
- NP does *NOT* stand for non-polynomial! It has not been proved that there are no P time algorithms for this class of problems, although it is believed there are none.





Class NP (cont.)

- Example: a **Hamiltonian path** in a directed graph G is a directed path that goes through each node exactly once.
- Easy to find exponential time algorithms for *HAMPATH*.
- Again: it is unknown whether there is a P time algorithm.



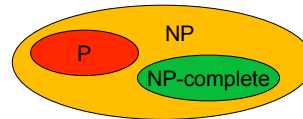
NP-Complete Problems

- Informal: **NP-complete problems** are the "hardest" problems in NP, i.e., they are the ones that are most likely not in P.
- A language L is NP-complete if it satisfies two conditions:
 - L is in NP, and
 - every L' in NP is polynomial time reducible to L .
- If someone could solve an NP-complete problem in polynomial time, that algorithm could be used to solve **all** NP-complete problems efficiently.
- Example:
 - satisfiability problem, SAT, $\phi = (x \text{ AND } y) \text{ or } (x \text{ OR NOT}(y))$
 - HAMPATH is NP-complete



NP-Hard Problems

- Informally: **NP-hard problems** is a class of decision problems that contains the problems that are at least as hard as any problem in NP, even though they may not be in NP themselves.
- NP-hard problems are NP-complete or harder.
- Formally: A language L is NP-hard, if $\forall L' \in \text{NP}, L' \leq_p L$.
- If L is also in NP, the L is **NP-complete**.



Space Complexity

- Informal: Compared to time complexity, where one is interested in the number of time steps, **space complexity** is concerned with the quantity of memory (or resources) it requires to solve a problem.
- More formal: Let M be a deterministic Turing machine that halts on all inputs. The **space complexity** of M is the function $f: \mathbf{N} \rightarrow \mathbf{N}$, where $f(n)$ is the maximum number of tape cells that M scans on any input of length n .
- Time and space are the most important considerations when looking for practical solutions and there are lots of associated tradeoffs.
- Computation in space is less popular than computation in time. (E.g., cellular automata versus von Neumann architectures.)



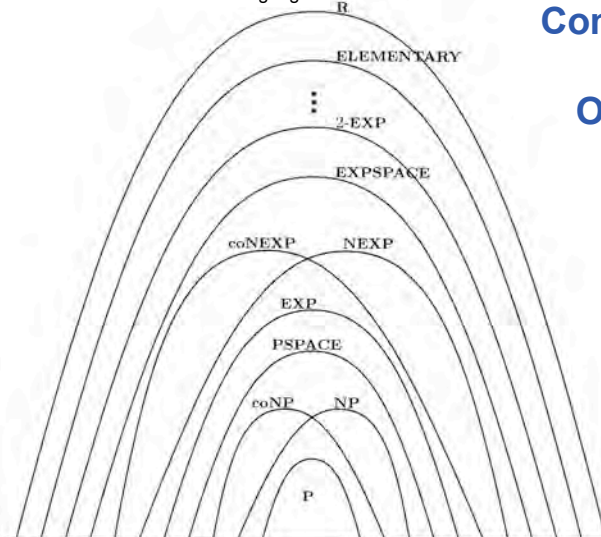
Space Complexity (cont.)

- **SPACE**($f(n)$) = $\{L \mid L \text{ is a language decided by an } O(f(n)) \text{ space deterministic Turing machine}\}$
- **NSPACE**($f(n)$) = $\{L \mid L \text{ is a language decided by an } O(f(n)) \text{ space non-deterministic Turing machine}\}$
- **PSPACE**: class of languages that are decidable in polynomial space on a deterministic Turing machine.
- Example: SAT can be solved with a linear space algorithm. Space is often more powerful than time because space can be reused!



Recursive languages

Complexity Classes Overview





Intractability

- Some problems, called **intractable**, require so much time and/or space that they cannot be solved in reality, although they are solvable in principle.
- Illustration:
 - Network with 100 on-off switches
 - 2^{100} possible configurations!
 - Testing one configuration takes $1\mu\text{s}$
 - $2^{100} \times 1\mu\text{s} \approx 4 \times 10^{16}$ years!!

