Why complexity theory?

- Complexity theory can answer questions on how easy or hard a problem is.
- Gives hints on what algorithms could be appropriate, e.g.:
  - Algorithms for **polynomial-time problems** are usually easy to design.
  - For **NP-complete** problems, backtracking and local search work well.
- Gives hints on what type of algorithm will (most probably) not work:
  - For problems that are believed to be harder than NP-complete ones, simple backtracking will not work.
- Gives hints on what sub-problems might be interesting.
Motivation
Basic
Notions: a Reminder
Algorithms and Turing machines
Problems, solutions, and complexity.
Complexity classes P and NP
Upper and lower bounds
Polynomial reductions
NP-completeness
Beyond NP
Oracle TMs and the Polynomial Hierarchy
Literature

Algorithms and Turing machines

- We use Turing machines as formal models of algorithms.
- This is justified, because:
  - we assume that Turing machines can compute all computable functions.
  - the resource requirements (in term of time and memory) of a Turing machine are only polynomially worse than other models.
- The regular type of Turing machine is the deterministic one: DTM (or simply TM).
- Often, however, we use the notion of nondeterministic TMs: NDTM.

October 28, 2013 Nebel, Wölfl, Hué – KRR
7 / 29

Problems, solutions, and complexity

- A problem is a set of pairs \((I,A)\) of strings in \(\{0,1\}^*\).
  - \(I\): instance; \(A\): answer
- If all answers \(A \in \{0,1\}\): decision problem
- A decision problem is the same as a formal language:
  - the set of strings formed by the instances with answer 1
- An algorithm solves (or decides) a problem if it computes the right answer for all instances.
- Complexity of an algorithm: function \(T : \mathbb{N} \rightarrow \mathbb{N}\),
  - measuring the number of basic steps (or memory requirement) the algorithm needs to compute an answer depending on the size of the instance.
- Complexity of a problem: complexity of the most efficient algorithm that solves this problem.

October 28, 2013 Nebel, Wölfl, Hué – KRR
8 / 29

Complexity classes P and NP

Problems are categorized into complexity classes according to the requirements of computational resources:

- The class of problems decidable on deterministic Turing machines in polynomial time: P
  - Problems in P are assumed to be efficiently solvable (although this might not be true if the exponent is very large).
  - In practice, this notion appears to be more often reasonable than not.
- The class of problems decidable on non-deterministic Turing machines in polynomial time: NP
- More classes are definable using other resource bounds on time and memory.

October 28, 2013 Nebel, Wölfl, Hué – KRR
9 / 29

Upper and lower bounds

- Upper bounds (membership in a class) are usually easy to prove:
  - provide an algorithm
  - show that the resource bounds are respected
- Lower bounds (hardness for a class) are usually difficult to show:
  - the technical tool here is the polynomial reduction (or any other appropriate reduction)
  - show that some hard problem can be reduced to the problem at hand.

October 28, 2013 Nebel, Wölfl, Hué – KRR
10 / 29
Polynomial reduction

- Given languages $L_1$ and $L_2$, $L_1$ can be polynomially reduced to $L_2$, written $L_1 \leq_p L_2$, if there exists a polynomially computable function $f$ such that
  $$ x \in L_1 \iff f(x) \in L_2. $$

  **Rationale:** it cannot be harder to decide $L_1$ than $L_2$
  - $L$ is hard for a class $C$ ($C$-hard) if all languages of this class can be reduced to $L$.
  - $L$ is complete for $C$ ($C$-complete) if $L$ is $C$-hard and $L \in C$.

NP-complete problems

- A problem is NP-complete iff it is NP-hard and in NP.
- Example: SAT (the satisfiability problem for propositional logic) is NP-complete (Cook/Karp)
  - Membership is obvious, hardness follows because computations on a NDTM correspond to satisfying truth-assignments of certain formulae

The complexity class co-NP

- Note that there is some asymmetry in the definition of NP:
  - It is clear that we can decide SAT by using a NDTM with polynomially bounded computation
  - There exists an accepting computation of polynomial length iff the formula is satisfiable
  - What if we want to solve UNSAT, the complementary problem?
    - It seems necessary to check all possible truth-assignments!
  - Define $co-C = \{ \Sigma^* \setminus L : L \subseteq \Sigma^* \text{ and } L \in C \}$ (provided $\Sigma$ is our alphabet)
  - $co-NP = \{ \Sigma^* \setminus L : L \subseteq \Sigma^* \text{ and } L \in NP \}$
  - Examples: UNSAT, TAUT $\in co-NP$!
  - **Note:** P is closed under complement, in particular,
    $$ P \subseteq NP \cap co-NP $$
PSPACE

There are problems even more difficult than NP and co-NP...

Definition ((N)PSPACE)

PSPACE (NPSPACE) is the class of decision problems that can be decided on deterministic (non-deterministic) Turing machines using only polynomially many tape cells.

Some facts about PSPACE:
- PSPACE is closed under complements (... as all other deterministic classes)
- PSPACE is identical to NPSPACE (because non-deterministic Turing machines can be simulated on deterministic TMs using only quadratic space)
- NP ⊆ PSPACE (because in polynomial time one can “visit” only polynomial space, i.e., NP ⊆ NPSPACE)
- It is unknown whether NP ⊈ PSPACE, but it is believed that

PSPACE-completeness

Definition (PSPACE-completeness)

A decision problem (or language) is PSPACE-complete if it is in PSPACE and all other problems in PSPACE can be polynomially reduced to it.

Intuitively, PSPACE-complete problems are the “hardest” problems in PSPACE (similar to NP-completeness). They appear to be “harder” than NP-complete problems from a practical point of view.

An example for a PSPACE-complete problem is the NDFA equivalence problem:

Instance: Two non-deterministic finite state automata \( A_1 \) and \( A_2 \).

Question: Are the languages accepted by \( A_1 \) and \( A_2 \) identical?

Other complexity classes ...

- There are complexity classes above PSPACE (EXPTIME, EXPSPACE, NEXPTIME, DEXPTIME ...)
- There are (infinitely many) classes between NP and PSPACE (the polynomial hierarchy defined by oracle machines)
- There are (infinitely many) classes inside P (circuit classes with different depths)
- ... and for most of the classes we do not know whether the containment relationships are strict

4 Oracle TMs and the Polynomial Hierarchy

- Oracle Turing machines
- Turing reduction
- Complexity classes based on OTMs
- QBF
Oracle Turing machines

- An Oracle Turing machine ((N)OTM) is a Turing machine (DTM, NDTM) with the possibility to query an oracle (i.e., a different Turing machine without resource restrictions) whether it accepts or rejects a given string.
- Computation by the oracle does not cost anything!
- Formalization:
  - a tape onto which strings for the oracle are written,
  - a yes/no answer from the oracle depending on whether it accepts or rejects the input string.
- Usage of OTMs answers what-if questions: What if we could solve the oracle-problem efficiently?

Turing reductions

- OTMs allow us to define a more general type of reduction
  - Idea: The “classical” reduction can be seen as calling a subroutine once.
  - \( L_1 \) is Turing-reducible to \( L_2 \), symbolically \( L_1 \leq_T L_2 \), if there exists a poly-time OTM that decides \( L_1 \) by using an oracle for \( L_2 \).
  - Polynomial reducibility implies Turing reducibility, but not vice versa!
  - NP-hardness and co-NP-hardness with respect to Turing reducibility are equivalent!
  - Turing reducibility can also be applied to general search problems!

Complexity classes based on Oracle TMs

- \( \text{P}^{\text{NP}} \) = decision problems solved by poly-time DTM with an oracle for a decision problem in NP.
- \( \text{NP}^{\text{NP}} \) = decision problems solved by poly-time NDTM with an oracle for a decision problem in NP.
- \( \text{co-NP}^{\text{NP}} \) = complements of decision problems solved by poly-time NDTM with an oracle for a decision problem in NP.
- \( \text{NP}^{\text{NP}} \) = ... 
  - ... and so on

Example

Consider the Minimum Equivalent Expression (MEE) problem:

- **Instance**: A well-formed Boolean formula \( \varphi \) using the standard connectives (not \( \leftrightarrow \)) and a non-negative integer \( k \).
- **Question**: Is there a well-formed Boolean formula \( \varphi' \) that contains \( k \) or fewer literal occurrences and that is logically equivalent to \( \varphi \)?

- This problem is NP-hard (wrt. to Turing reductions).
- It does not appear to be NP-complete.
- We could guess a formula and then use a SAT-oracle ...
- MEE \( \in \text{NP}^{\text{NP}} \).
The polynomial hierarchy

The complexity classes based on OTMs form an infinite hierarchy.

The polynomial hierarchy \( \text{PH} \)

\[
\begin{align*}
\Sigma^p_0 &= P \\
\Sigma^p_{i+1} &= \text{NP} \cap \Sigma^p_i \\
\Pi^p_0 &= P \\
\Pi^p_{i+1} &= \text{co-NP} \cap \Pi^p_i \\
\Delta^p_0 &= P \\
\Delta^p_{i+1} &= \text{PSPACE} \\
\end{align*}
\]

- \( \text{PH} = \bigcup_{i \geq 0} (\Sigma^p_i \cup \Pi^p_i \cup \Delta^p_i) \subseteq \text{PSPACE} \)
- \( \text{NP} = \Sigma^p_1 \)
- \( \text{co-NP} = \Pi^p_1 \)

Quantified Boolean formulae: definition

The evaluation problem of QBF generalizes both the satisfiability and validity/tautology problems of propositional logic. The latter are \( \text{NP} \)-complete and \( \text{co-NP} \)-complete, resp., whereas the former is \( \text{PSPACE} \)-complete.

Example
The formulae \( \forall x \exists y(x \leftrightarrow y) \) and \( \exists x \forall y(x \land y) \) are true.

Example
The formulae \( \exists x \forall y(x \leftrightarrow y) \) and \( \forall x \exists y(x \lor y) \) are false.

October 28, 2013
Nebel, Wölfl, Hué – KRR 29 / 29

Quantified Boolean formulae: definition

The evaluation problem of QBF generalizes both the satisfiability and validity/tautology problems of propositional logic. The latter are \( \text{NP} \)-complete and \( \text{co-NP} \)-complete, resp., whereas the former is \( \text{PSPACE} \)-complete.

Example
The formulae \( \forall x \exists y(x \leftrightarrow y) \) and \( \exists x \forall y(x \land y) \) are true.

Example
The formulae \( \exists x \forall y(x \leftrightarrow y) \) and \( \forall x \exists y(x \lor y) \) are false.

October 28, 2013
Nebel, Wölfl, Hué – KRR 27 / 29

The Polynomial Hierarchy: connection to QBF

Truth of QBFs with prefix \( \exists \forall \ldots \) is \( \Pi^p_1 \)-complete.

Truth of QBFs with prefix \( \forall \exists \ldots \) is \( \Sigma^p_1 \)-complete.

Special cases corresponding to \( \text{SAT} \) and \( \text{TAUT} \):
- The truth of QBFs with prefix \( \exists x_1 \ldots x_n \) is \( \text{NP} = \Sigma^p_1 \)-complete.
- The truth of QBFs with prefix \( \forall x_1 \ldots x_n \) is \( \text{co-NP} = \Pi^p_1 \)-complete.
Literature

M. R. Garey and D. S. Johnson.
Computers and Intractability – A Guide to the Theory of
NP-Completeness.

C. H. Papadimitriou.
Computational Complexity.