Declarative Knowledge Processing

Brief Recap of Computational Complexity

Magdalena Ortiz

Knowledge Base Systems Group
Institute of Information Systems
ortiz@kr.tuwien.ac.at

10 October 2012

Outline

1. Basic definitions
2. Hardness and completeness
   2.1 Reductions
   2.2 Hardness and completeness
3. Most important complexity classes
   3.1 Tractability and intractability: P and NP
   3.2 Complexity classes above NP
   3.3 Complexity classes below P
   3.4 Relationship between the complexity classes

Problems

Definition of Problem

A problem $P$ is a question together with a (in general infinite) set of instances (i.e. possible inputs).

- We assume each instance is encoded in a meaningful (compact) way (i.e., as a compact string)

$P$ is a decision problem if the question is has a yes/no answer

Example Problem: REACHABILITY

INSTANCE: A graph $(V, E)$ and nodes $u, v \in V$.
QUESTION: Is there a path in the graph from $u$ to $v$?

REACHABILITY is a decision problem

Algorithms

Definition of Algorithm

An algorithm for a problem $P$ is a description of computation steps that allow to solve any given instance of the problem $P$.

- An algorithm performs finitely many simple computation steps.

- It works on all instances of the problem: for each possible instance of the problem, the execution of the computation steps terminates and gives the correct answer to the question.
  - input: an instance of the problem
  - output: yes or no
Complexity Theory

Complexity theory aims at understanding how difficult it is to solve specific problems.

- Standard complexity theory deals (only) with decision problems.
- The difficulty (complexity) is measured in terms of the amount of resources that the (best possible) algorithm needs to solve the problem.
- We consider two main resources:
  - time
  - space

Measuring the computational complexity

- **Worst-case** complexity analysis: the complexity is measured in terms of a (complexity) function $f$:
  - argument: the size $n$ of an instance of the problem (i.e., the length of its encoding)
  - result: the amount $f(n)$ of time/space needed in the worst-case to solve an instance of size $n$
- The asymptotic behavior of the complexity function when $n$ grows is considered.
- To abstract away from contingent issues (e.g., programming language, processor speed, etc.), we always refer to an abstract computing model: Turing Machines (TMs).

Complexity classes

Usually one does not consider specific complexity functions $f$, but rather families $C$ of complexity functions.

- This makes complexity theory robust w.r.t. different encodings.

Each family of functions gives rise to a complexity class.

**Definition of Complexity Class**

A time/space complexity class $C$ is the set of all problems $P$ such that an instance of $P$ of size $n$ can be solved in time/space at most $C(n)$.

**Note:** Once we have fixed and an encoding of the instances of a decision problem $P$ into strings over some alphabet, we can see the problem as the language $L_P$ that contains precisely (the strings encoding) those instances for which the answer is yes.

Technically, a complexity class is actually a set of languages.

Reductions

To establish lower bounds on the complexity of problems, we make use of the notion of reduction.

**Definition of Reduction**

A reduction from a problem $P_1$ to a problem $P_2$ is a function $R$ (the reduction) from instance of $P_1$ to instances of $P_2$ such that:

1. $R$ is efficiently computable (i.e., in logarithmic space), and
2. An instance $I$ of $P_1$ has answer yes iff $R(I)$ has answer yes.

$P_1$ reduces to $P_2$ if there is a reduction $R$ from $P_1$ to $P_2$.

**Intuition:** If $P_1$ reduces to $P_2$, then $P_2$ is at least as difficult as $P_1$, since we can solve an instance $I$ of $P_1$ by reducing it to the instance $R(I)$ of $P_2$ and then solve $R(I)$. 
Hardness and completeness

Definition of Hardness
A problem $P$ is hard for a complexity class $C$ if every problem in $C$ reduces to $P$.

Hence, if we can solve a problem that is hard for $C$, we can solve all problems in $C$.

Definition of Completeness
A problem $P$ is complete for a complexity class $C$ if
1. it belongs to $C$, and
2. it is hard for $C$.

Intuitively, a problem that is complete for $C$ is among the hardest problems in $C$.

Tractability and intractability: $P$ and $NP$

$P$ (sometimes $PTime$)
Set of problems solvable in polynomial time by a deterministic TM.
- These problems are considered tractable, i.e., solvable for large inputs.

$NP$
Set of problems solvable in polynomial time by a non-deterministic TM.
- These problems are believed intractable, i.e., unsolvable for large inputs.
- The best known algorithms actually require exponential time.
- For $NP$ problems, the following type of algorithm can be used:
  1. Non-deterministically guess a possible solution of polynomial size.
  2. Check in polynomial time that the guessed solutions is good.

NP and co-NP
The complement of a decision problem $P$ is obtained by inverting its question.

Example: the complement of $REACHABILITY$

Problem: $UNREACHABILITY$
INSTANCE: A graph $(V, E)$ and nodes $u, v \in V$.
QUESTION: Is there no path in the graph from $u$ to $v$?

coNP
Set of problems whose complement is in $NP$.
- $NP$ and $co-NP$ capture large classes of practical problems
- For many of them, good heuristics and practicable algorithms have been found

To think: Can we use guess-and-check algorithms as above for co-$NP$ problems?

Complexity classes above $NP$

PSPACE
Set of problems solvable in polynomial space by a deterministic TM.
- These problems may require exponential time.
- They are at least as difficult than $NP$ problems, and in fact, they are believed to be more difficult.
- Practical algorithms and heuristics work less well than for $NP$ problems.
- The polynomial hierarchy is a hierarchy of complexity classes at least as hard as $NP$, but not harder than $PSPACE$
  - This hierarchy is believed to be infinite
  - Two classes in the hierarchy:
    - $\Sigma_P^P$: problems solvable in $NP$ using an $NP$-oracle
    - $\Pi_P^P$: problems solvable in co-$NP$ using an $NP$-oracle
Above PSPACE

**ExpTime**

Set of problems solvable in exponential time by a deterministic TM.

- This is the first provably intractable complexity class.
- These problems are considered to be very difficult.

**NEXPTIME**

Set of problems solvable in exponential time by a non-deterministic TM.

**2ExpTime**

Set of problems solvable in double exponential time (i.e. time bounded asymptotically by $2^{n^n}$) by a deterministic TM.

2NEXPTIME, 3ExpTime, 3NEXPTIME, ... are defined analogously

Complexity classes below P

**L and NL (sometimes LogSpace and NLogSpace )**

Set of problems solvable in logarithmic space by a (non-)deterministic TM.

- When measuring the space complexity, the size of the input does not count and only the working memory (TM tape) is considered.
- Note that in logarithmic space we can not even load the full input into the working memory.
- Logarithmic space computations compose (this is not trivial).
- Correspond to reachability in undirected and directed graphs, respectively.
- NL = coNL

Complexity classes below P, part 2

**AC 0**

Set of problems solvable in constant time using a polynomial number of processors.

- These problems are solvable efficiently even for very large inputs.
- Corresponds to the complexity of model checking a fixed FO formula when the input is the model only.

Relationship between the complexity classes

The following relationships are known:

$$AC_0 \subseteq L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE \subseteq \text{ExpTime}$$

We also know that

- $P \subseteq \text{ExpTime} \subseteq 2\text{ExpTime}$
- Deterministic classes are closed under complement
- $PSPACE = \text{NPSPACE}$ (Savitch’s theorem)
- (this also holds for higher space classes, like EXPSPACE)