Limits of Feasibility

Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria
http://www.risc.jku.at

Complexity Relationships among Models

All Turing complete models can perform the “same” computations; but the complexities of these computations may be different.

- Different notions of time and space consumption.
  - Functions $t(i)$ and $s(i)$.
- Different input encodings with different sizes.
  - Function $|i|$.

“Same” computation performed in different models may be even in different complexity classes.

Example

Given $n \in \mathbb{N}$, compute $r := 2^{2^n}$.

```
$r := 2;\ i := 0$
**while** $i < n$ **do**
    $r := r * r;\ i := i + 1$
```

We analyze the complexity on a Random Access Machine.

- Space complexity: number of registers used.

$$S_1(n) := O(1)$$

- Time complexity: number of instructions executed.
  - We assume new instruction `MUL (r)` to multiply accumulator with content of register r.

$$T_1(n) := \Theta(n)$$

We have assumed that every register can hold an arbitrarily large number and multiplication can be performed in a single step.

Example (Contd)

But in the original RAM model there was only an instruction `ADD #c` to add to the accumulator a constant c.

- Computation of $n+n$:

$$\text{Time } \Theta(n)$$

- Computation of $n*n$:

$$\text{Time } \Theta(n^2)$$

- Computation of $2^{2^{n-1}} * 2^{2^{n-1}}$:
  - Last multiplication in algorithm.

$$\text{Time } \Theta(2^{2^n})$$

- Time complexity of computation:

$$T_2(n) = \Omega(2^{2^n})$$
$$T_2(n) = O(n \cdot 2^{2^n})$$

$T_1(n)$ and $T_2(n)$ are in vastly different complexity classes.

Example (Contd)

- Problem: we performed analysis with "unrealistic" cost models.
  - First model too much demanding: multiplication of arbitrarily large numbers in a single step.
  - Second model too little demanding: only addition by a constant in a single step.
- Solution: a "realistic" cost model for RAM computations.
  - Number n is represented by $l(n) := 1 + \lfloor \log_b n \rfloor = \Theta(\log n)$ digits.
  - RAM can perform arithmetic on individual digits in time $O(1)$.
  - Realistic space complexity: the digit length of a number.

$$|n| = \Theta(\log n)$$

  - Realistic time complexity: the number of digit operations.
    - $n+n$: time $\Theta(\log n)$.
    - $n*n$: time $O((\log n)^2)$ (no tight bound known).

We are going to use the realistic RAM cost model.

Example (Contd)

Complexity of $r := 2^{2^n}$ in the realistic RAM cost model.

- Largest number stored in r is 2^{2^n}.

$$S(n) = \Theta(\log 2^{2^n}) = \Theta(2^n)$$

- Sequence of n multiplications:
  - $O((\log 2)^2 + (\log 4)^2 + \ldots + (\log 2^{2^{n-1}})^2) = O(1^2 + 2^2 + \ldots + (2^{n-1})^2) = O(\sum_{i=0}^{n-1} (2^i)^2)$
  - $\sum_{i=0}^{n-1} (2^i)^2 = \frac{4^n - 1}{3}$

$$T(n) = O(4^n)$$

- Comparison with the unrealistic models:

$$S_1(n) = o(S(n))$$
$$T_1(n) = o(T(n)) = o(T_2(n))$$

In the realistic cost model, the computation needs more space; the required time is between the previously established bounds.

Logarithmic Cost Model of RAM

We are formalizing the "realistic" RAM cost model.

- The logarithmic cost model of a RAM assigns to every number n stored on the input tape or in a register the size

$$|n| = \Theta(\log n)$$

- Space complexity: the maximum, at any time during the computation, of the sums $\sum_n |n|$ of the sizes of all numbers stored in the RAM.
- Time complexity: the sums of the execution times of all instructions executed by the RAM:
  - every instruction involving a number n on a tape cell or in a register is assigned time $|n|$;
  - every other instruction is assigned time 1.

A useful approximations for the resource consumption of a real computer.

Complexity of RAM versus Turing Machine

- Theorem: every computation of a Turing machine with time $O(T(n))$ can be simulated by a Random Access Machine (under the logarithmic cost model) with time

$$O(T(n) \cdot \log T(n))$$

- Theorem: every computation of a Random Access Machine with time $O(T(n))$ (under the logarithmic cost model) can be simulated by a Turing machine with time

$$O(T(n)^4)$$

  - Proofs by analysis of the presented simulations.

The time complexities of the same computations in both models differ by not more than a "polynomial transformation" (a polynomial time complexity in one model remains polynomial in the other one).

Invariance Thesis

Cees F. Slot and Peter van Emde Boas, 1984.

Invariance Thesis: "reasonable" machines (Turing complete computational models) can simulate each other within a polynomially bounded overhead in time and a constant-factor overhead in space.

Consequence: a polynomial time computation in one Turing complete model can be performed in polynomial time also in *any* other such model.

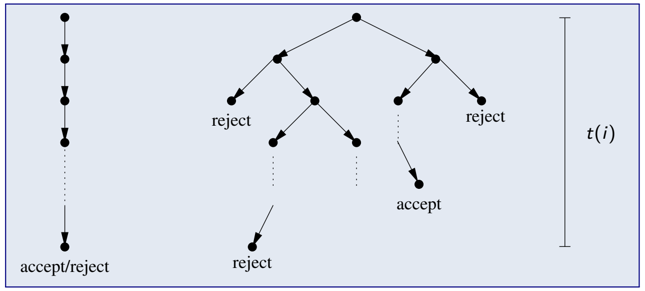
- Not a provable theorem, but an unprovable thesis.
- Validated many times for different Turing complete models.
  - Possible exception: quantum computing (integer factorization can be performed in polynomial time on a quantum computer, but no polynomial time algorithm for a classical model is known).

An indication for a fundamental border between those computations that can be performed in polynomial time and those that cannot.

Non-Deterministic Turing Machines

Nondeterministic Turing machine M.

- Time consumption $t(i)$: maximum number of configurations for any run of M with input i.
- Space consumption $s(i)$: distance from the beginning of the tape for any run of M with input i.



Since M can simultaneously investigate multiple branches, we consider the time of the longest one (alternative: the shortest accepting one).

Example

Hamiltonian Cycles Problem: given an undirected graph G with n nodes, does there exist a cyclic path in G that visits every node exactly once?

G can be represented on the tape by a sequence of $b = n^2$ bits where bit $i \cdot n + j$ indicates whether node i is connected to node j.

- Deterministic M: best solution known requires exponential time.
- Nondeterministic M: can solve problem in polynomial time.
  1. M writes non-deterministically n numbers from $1 \dots n$ to the tape.
     - Can be performed in a time which is polynomial in b.
  2. M checks whether the n numbers represent a Hamiltonian cycle in G.
     - Can be performed in a time which is polynomial in b.

  If there is any cyclic path in G, it will be detected by some execution branch of M in polynomial time.

A deterministic Turing machine needs to construct a solution; a nondeterministic one can "guess" and "check".

Problem Complexity

- Problem Complexity: A decidable problem P has (non)deterministic time complexity $T(n)$ respectively space complexity $S(n)$, if there exists a (non)deterministic Turing machine M that decides P, and
  - for every input w with length $n = |w|$,
  - M terminates in time $O(T(n))$ and uses space $O(S(n))$.
- Problem Complexity Classes:

$$\begin{aligned}
DTIME(T(n)) &:= \{P \mid P \text{ has deterministic time complexity } T(n)\}\\
NTIME(T(n)) &:= \{P \mid P \text{ has nondeterministic time complexity } T(n)\}\\
DSPACE(T(n)) &:= \{P \mid P \text{ has deterministic space complexity } T(n)\}\\
NSPACE(T(n)) &:= \{P \mid P \text{ has nondeterministic space complexity } T(n)\}
\end{aligned}$$

Classes of decidable problems with a specific deterministic/nondeterministic time/space complexity.

Problem Complexity Classes

See `https://complexityzoo.uwaterloo.ca/Complexity_Zoo` for almost 500 problem complexity classes.

$$\begin{aligned}
\mathcal{P} &:= \bigcup_{i \in \mathbb{N}} DTIME(n^i)\\
\mathcal{NP} &:= \bigcup_{i \in \mathbb{N}} NTIME(n^i)\\
PSPACE &:= \bigcup_{i \in \mathbb{N}} DSPACE(n^i)\\
NSPACE &:= \bigcup_{i \in \mathbb{N}} NSPACE(n^i)\\
EXPTIME &:= \bigcup_{i \in \mathbb{N}} DTIME(2^{n^i})\\
NEXPTIME &:= \bigcup_{i \in \mathbb{N}} NTIME(2^{n^i})
\end{aligned}$$

Classes of decidable problems that can be solved with deterministic/nondeterministic polynomial/exponential time/space.

Cobham's Thesis

Alan Cobham, 1965.

Cobham's Thesis: A problem is "tractable", i.e., can be feasibly decided, if it is in complexity class $\mathcal{P}$.

- A crude rule of thumb of what can be "reasonably" computed.

If we accept Cobham's Thesis and also the Invariance Thesis, the class of tractable problems is the same for every computational model.

Solving versus Verifying

- Verifier for a language L: a Turing machine V such that

$$L = \{w \mid V \text{ accepts } \langle w, c\rangle \text{ for some word } c\}$$

  - For every word w in L, there is some extra input c (the certificate) that drives the execution of V towards acceptance of w.
  - A polynomial time verifier runs in polynomial time in the length of w.
- Theorem: $L \in \mathcal{NP}$ iff there exists a polynomial time verifier for L.
  - $\Rightarrow$ Take non-deterministic M that decides L in polynomial time. Construct V that for input $\langle w, c\rangle$ simulates the execution of M on w by interpreting c as the sequence of non-deterministic choices to be made.
  - $\Leftarrow$ Take verifier V for L that runs in polynomial time $T(n)$. Construct M that for input w first non-deterministically creates every possible certificate c of length up to $T(n)$ and then simulates V on $\langle w, c\rangle$.

Solving a problem non-deterministically in polynomial time is equivalent to verifying the solution deterministically in polynomial time.

Relationships between Complexity Classes

Theorems:

1. $\mathcal{P} \subseteq \mathcal{NP}$
2. $EXPTIME \subseteq NEXPTIME$
3. $\mathcal{P} \subseteq PSPACE$
4. $\mathcal{NP} \subseteq NSPACE$
5. $PSPACE \subseteq EXPTIME$

Proofs:

- 1. and 2. follow from the fact that every deterministic Turing machine is a special case of a nondeterministic one.
- 3. and 4. follow from the fact that every Turing machine, in order to consume one unit of space, has to write one cell to the tape, i.e., it consumes one unit of time. A Turing machine therefore cannot consume more memory than time.
- 5. follows from the fact that, if the space of a Turing machine is constrained by a polynomial bound $O(n^i)$, then the number of configurations of the Turing machine is constrained by an exponential bound $O(2^{n^i})$; consequently the Turing machine cannot make more than exponentially many steps until it halts.

Relationships between Complexity Classes

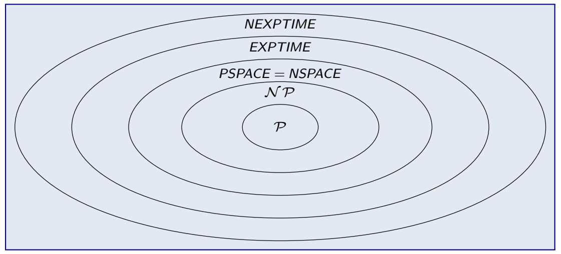
Walter Savitch, 1970.

Savitch's Theorem: every problem that is decidable by a nondeterministic Turing machine with space $O(S(n))$ is also decidable by a deterministic Turing machine with space $O(S(n)^2)$:

$$NSPACE(S(n)) \subseteq DSPACE(S(n)^2)$$

Consequently, $NSPACE = PSPACE$.

Relationships between Complexity Classes



$$\boxed{\mathcal{P} \subseteq \mathcal{NP} \subseteq PSPACE = NPSPACE \subseteq EXPTIME \subseteq NEXPTIME}$$

- $\mathcal{P} \neq EXPTIME$ and
- $\mathcal{NP} \neq NEXPTIME$

At least one of the first three subset relationships and at least one of the last three subset relationships must be a proper one.

$\mathcal{P} \stackrel{?}{=} \mathcal{NP}$

The most famous open problem in theoretical computer science.

$$\mathcal{P} \stackrel{?}{=} \mathcal{NP}$$

- Can every problem that can be solved in polynomial time by a nondeterministic Turing machine also be solved in polynomial time by a deterministic one?
- Is it not more difficult (time consuming) to "construct" a solution to a problem than to "guess" and subsequently "check" it?
  - Intuitively, the answer to these questions is "no", i.e, $\mathcal{P} \subsetneq \mathcal{NP}$.
  - The Clay Mathematics Institute will pay US\$$1,000,000$ for a correct proof of either $\mathcal{P} = \mathcal{NP}$ or $\mathcal{P} \neq \mathcal{NP}$.

While most people believe $\mathcal{P} \subsetneq \mathcal{NP}$, the answer is still unknown.

Polynomial-Time-Reducibility

For investigating $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$, it is useful to compare problems according to how time-consuming they are to solve.

- A decision problem $P \subseteq \Sigma^*$ is polynomial time-reducible to a decision problem $P' \subseteq \Gamma^*$ ($P \leq_P P'$):
  - there is a function $f : \Sigma^* \to \Gamma^*$ such that for all $w \in \Sigma^*$

$$P(w) \Leftrightarrow P'(f(w))$$

  - and f can be computed by a deterministic Turing machine in polynomial time.

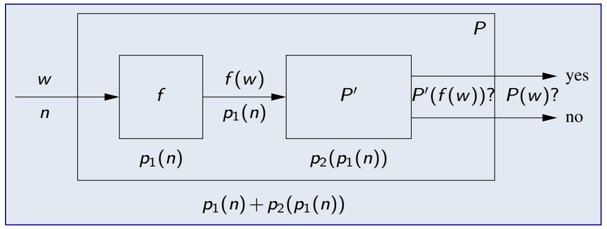
If $P \leq_P P'$, then a decision of P is "essentially" (up to a polynomial transformation) not more time-consuming than a decision of P'.

Polynomial Time-Reducibility and $\mathcal{P}/\mathcal{NP}$

- Theorem: for all decision problems P and P' with $P \leq_P P'$, we have

$$P' \in \mathcal{P} \Rightarrow P \in \mathcal{P}$$
$$P' \in \mathcal{NP} \Rightarrow P \in \mathcal{NP}$$

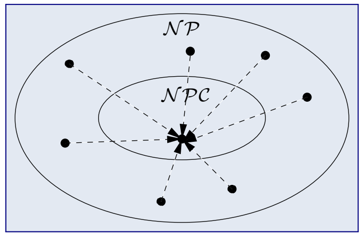
  - If P is polynomial time-reducible to P' and P' can be (non)deterministically decided in polynomial time, then also P can be (non)deterministically decided in polynomial time.
- Proof: for deciding $P(w)$, input w of size n can be transformed in time $p_1(n)$ to word $f(w)$ of size $p_1(n)$. $P'(f(w))$ can be decided in time $p_2(p_1(n))$; the decision of $P(w)$ thus takes time $p_1(n) + p_2(p_1(n))$.



$\mathcal{NP}$-Completeness

- A problem P' in complexity class $\mathcal{NP}$ is $\mathcal{NP}$-complete, if for every problem $P \in \mathcal{NP}$ we have $P \leq_P P'$.

$$\mathcal{NPC} := \{P' \in \mathcal{NP} \mid P' \text{ is } \mathcal{NP}\text{-complete}\}$$



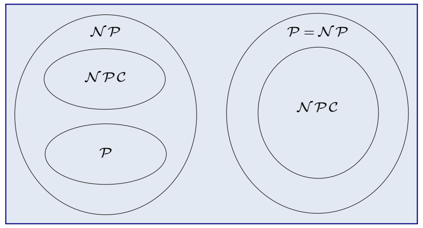
Every problem in $\mathcal{NP}$ can be reduced in polynomial time to any problem in $\mathcal{NPC}$.

$\mathcal{NP}$-Completeness and $\mathcal{P} \overset{?}{=} \mathcal{NP}$

- Theorem: if $\mathcal{NPC} \cap \mathcal{P} = \emptyset$, then $\mathcal{P} \neq \mathcal{NP}$, and vice versa.

  If no $\mathcal{NP}$-complete problem can be deterministically decided in polynomial time, then $\mathcal{P} \neq \mathcal{NP}$, and vice versa.
- Theorem: if $\mathcal{NPC} \cap \mathcal{P} \neq \emptyset$, then $\mathcal{NPC} \subseteq \mathcal{P}$.

  If some $\mathcal{NP}$-complete problem can be deterministically decided in polynomial time, then all such problems can.
- Consequence:
  1. either $\mathcal{NPC} \cap \mathcal{P} = \emptyset$; then $\mathcal{P} \neq \mathcal{NP}$ (and also $\mathcal{NPC} \neq \mathcal{NP}$);
  2. or $\mathcal{NPC} \cap \mathcal{P} \neq \emptyset$; then $\mathcal{NPC} \subseteq \mathcal{P} = \mathcal{NP}$.



Core question: does there exist some $\mathcal{NP}$-complete problem that is deterministically decidable in polynomial time or not?

Cook's Theorem

Does there exist any $\mathcal{NP}$-complete problem at all?

- Propositional formula F:

$$F ::= x_i \mid \neg F \mid F \wedge F \mid F \vee F$$

- Satisfiability Problem:
  - Is F satisfiable, i.e., does there exist an assignment of the variables x_i in F to truth values which makes F true?
    - $(x_1 \vee x_2) \wedge (\neg x_2 \vee x_1)$ is satisfiable by $x_1 := \mathrm{T}, x_2 := \mathrm{F}$.
    - $x_1 \wedge \neg x_2 \wedge (\neg x_1 \vee x_2)$ is not satisfiable.
- Cook's Theorem: the satisfiability problem is $\mathcal{NP}$-complete.
  - Stephen Cook, 1971.

Other problems can be shown to be $\mathcal{NP}$-complete by proving that the satisfiability problem is polynomial time-reducible to these problems.

$\mathcal{NP}$-Complete Problems

Currently more than 3000 problems from widely different mathematical areas are known to be $\mathcal{NP}$-Complete.

- Hamiltonian Path: already discussed.
- Traveling Salesman: find in a weighted graph the shortest cyclic path that connects all nodes.
- Graph Coloring: decide whether the nodes of graph G can be colored with n colors such that no adjacent nodes get the same color.
- Knapsack: decide, given a collection of items with a certain "weight" and "value", which subset is the most valuable one among those that can be packed into a "knapsack" with a certain weight limit.
- Integer Programming: given integer matrix A and vectors b, c, determine that vector x with $A \cdot x \leq b$ that maximizes $c \cdot x$.

All are polynomial time-reducible to the satisfiability problem which can be tackled by (heuristically fast) "SAT solvers".

Complements of Problems

The quest for $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$ can be also tackled by considering "co-problems".

- Theorem: for all decision problems P and P' with $P \leq_P P'$, we have

$$\overline{P} \leq_P \overline{P'}$$

  - If P is pol. time-reducible to P', than also the co-problem $\overline{P}$ is to $\overline{P'}$.
  - Proof: we assume $P \leq_P P'$ and show $\overline{P} \leq_P \overline{P'}$. We have to find a function f that can be computed by a deterministic Turing machine in polynomial time such that $\overline{P}(w) \Leftrightarrow \overline{P'}(f(w))$ which is equivalent to $P(w) \Leftrightarrow P'(f(w))$. From $P \leq_P P'$, we have exactly such an f.
- Theorem: problem P is in $\mathcal{P}$, if and only if its co-problem $\overline{P}$ is in $\mathcal{P}$:

$$\mathcal{P} = \{P \mid \overline{P} \in \mathcal{P}\}$$

  - We can decide by a deterministic Turing machine in polynomial time whether some input satisfies P, if and only if we can decide by such a machine in polynomial time whether some input does *not* satisfy P.
  - Proof: if a deterministic machine can decide in polynomial time P, by reverting its answer it can decide in polynomial time $\overline{P}$, and vice versa.

co-$\mathcal{NP}$

- co-$\mathcal{NP}$: the class of all problems whose complements are in $\mathcal{NP}$:

$$\text{co-}\mathcal{NP} := \{P \mid \overline{P} \in \mathcal{NP}\}$$

  - For every $P \in$ co-$\mathcal{NP}$ a nondeterministic Turing machine can decide in polynomial time whether some input is *not* in P.
  - If P can be nondeterministically decided in polynomial time, this does not necessarily imply that also $\overline{P}$ can be decided so:
    - M accepts w, if there is at least one accepting run with answer "yes". There may be also runs with answer "no" that do not accept w.
    - If $\overline{M}$ just reverts the "yes" and "no" answers from M, then also $\overline{M}$ may accept w, but then $L(\overline{M}) \neq \overline{P}$.
  - Thus $\mathcal{NP} \neq$ co-$\mathcal{NP}$ is possible.

To decide $\overline{P}$, $\overline{M}$ may accept w only, if M does for input w not have *any* run with a "yes" answer; since the number of runs of M may be exponential in $|w|$, it is unclear how $\overline{M}$ can do this in polynomial time.

$\mathcal{P}$ versus $\mathcal{NP}$ and co-$\mathcal{NP}$

- Theorem: $\mathcal{P} \subseteq \mathcal{NP} \cap$ co-$\mathcal{NP}$
  - For every problem in $\mathcal{P}$, both the problem and its complement can be decided by a nondeterministic Turing machine in polynomial time.
  - Proof: we take arbitrary $P \in \mathcal{P}$ and show $P \in \mathcal{NP} \cap$ co-$\mathcal{NP}$. Since $P \in \mathcal{P}$, also $P \in \mathcal{NP}$. It thus suffices to show $P \in$ co-$\mathcal{NP}$. We know $\overline{P} \in \mathcal{P}$. Since $\mathcal{P} \subseteq \mathcal{NP}$, we know $\overline{P} \in \mathcal{NP}$ and thus $P \in$ co-$\mathcal{NP}$.
- Theorem: If $\mathcal{NP} \neq$ co-$\mathcal{NP}$, then $\mathcal{P} \neq \mathcal{NP}$.
  - Proof: we assume $\mathcal{NP} \neq$ co-$\mathcal{NP}$ and show $\mathcal{P} \neq \mathcal{NP}$. We define the property $C(S)$ of a class of problems S as

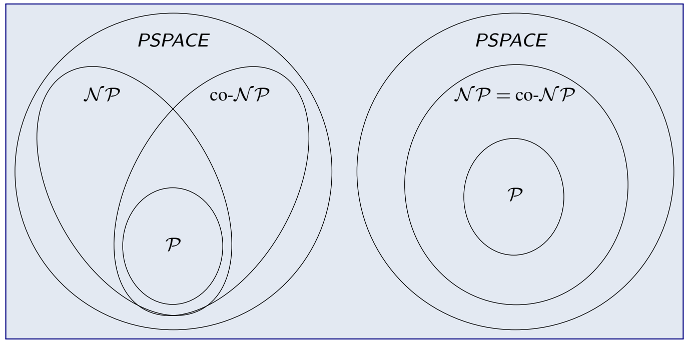
$$\mathcal{C}(S) :\Leftrightarrow S = \{P \mid \overline{P} \in S\}$$

    We know $C(\mathcal{P})$. From $\mathcal{NP} \neq$ co-$\mathcal{NP}$, we know $\neg C(\mathcal{NP})$. Thus $\mathcal{P} \neq \mathcal{NP}$.

The class co-$\mathcal{NP}$ becomes relevant to the quest for $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$.

$\mathcal{P}$ versus $\mathcal{NP}$ and co-$\mathcal{NP}$

From the previous theorems, we have two possibilities.

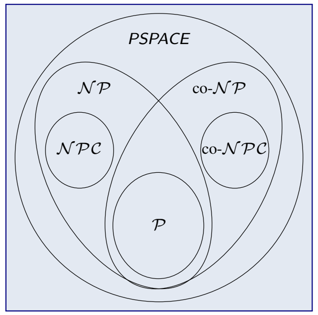


1. either $\mathcal{NP} \neq$ co-$\mathcal{NP}$ and $\mathcal{P} \subseteq \mathcal{NP} \cap$ co-$\mathcal{NP}$ and $\mathcal{P} \neq \mathcal{NP}$,
2. or $\mathcal{P} \subseteq \mathcal{NP} =$ co-$\mathcal{NP}$ ($\mathcal{P} \neq \mathcal{NP}$ may or may not hold).

Since also co-$\mathcal{NP} \subseteq$ *PSPACE*, all classes are contained in *PSPACE*.

co-$\mathcal{NP}$ versus $\mathcal{NPC}$

- Theorem: $\mathcal{NP} \neq$ co-$\mathcal{NP} \Leftrightarrow \forall P \in \mathcal{NPC} : \overline{P} \notin \mathcal{NP}$
  - $\mathcal{NP} \neq$ co-$\mathcal{NP}$, if and only if the complements of all $\mathcal{NP}$-complete problems cannot be nondeterministically decided in polynomial time.
  - Proof: see lecture notes.
- We thus can refine the first possibility of the previous diagram:



Most likely situation as conjectured by most researchers.

co-$\mathcal{NP}$ versus $\mathcal{NPC}$

- If there were a $P \in \mathcal{NPC}$ with $\overline{P} \in \mathcal{NP}$, then $\mathcal{P} = \mathcal{NP}$ might hold.
  - No such problem could be found yet.

    An indication that $\mathcal{P} \neq \mathcal{NP}$.
  - Actually, for no $P \in \mathcal{NP}$ it is even known whether $\overline{P} \in \mathcal{NP}$.
- If a problem can be shown to be both in $\mathcal{NP}$ and in co-$\mathcal{NP}$, this may be considered as evidence that the problem is *not* $\mathcal{NP}$-complete.

  Because otherwise $\mathcal{P} = \mathcal{NP}$ might hold.
- This is the case for the problem of “integer factorization”.

  Determine whether a given natural number can be decomposed into non-trivial prime factors.
  - Public-key cryptography assumes that this is a “hard” problem.

    No deterministic polynomial time algorithm is known.
  - But both the problem and its co-problem are nondeterministically decidable in polynomial time.

    Thus the problem might not be $\mathcal{NP}$-complete and consequentially not so “hard” as assumed.

Theoretical considerations with practical implications in computer security.

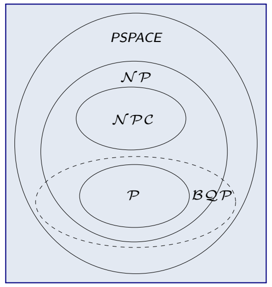
Quantum Computing

What about the emerging class of “quantum computers”?

- Complexity class $\mathcal{BQP}$ (bounded error quantum polynomial time).

  The class of problems solvable by a quantum computer in polynomial time with limited error probability.
- Integer factorization is in $\mathcal{BQP}$ (Peter Shor, 1994)
  - Quantum computers might break public key cryptography.
  - But integer factorization may not be $\mathcal{NP}$-complete.
- Theorem: $\mathcal{P} \subseteq \mathcal{BQP} \subseteq$ *PSPACE*
  - Conjecture: $\mathcal{NPC}$ and $\mathcal{BQP}$ are disjoint, $\mathcal{BQP}$ and $\mathcal{NP}$ are incomparable.

No quantum algorithm is known that solves any $\mathcal{NP}$-complete problem in polynomial time.


PSPACE
$\mathcal{NP}$
$\mathcal{NPC}$
$\mathcal{P}$
$\mathcal{BQP}$