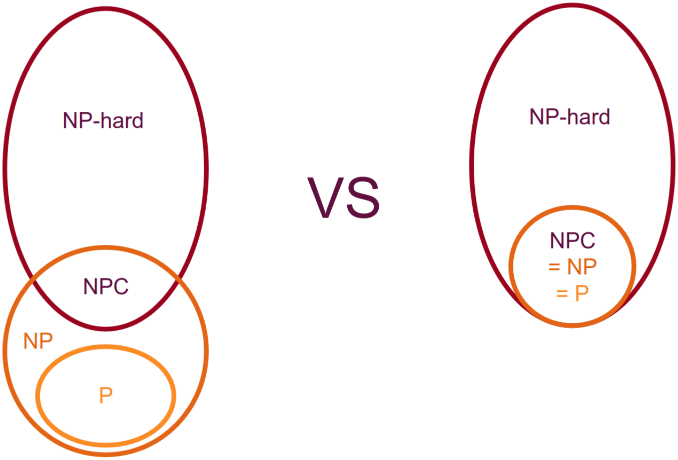
EITQ

Advanced Classical Complexity[1]

Renaud Vilmart

[1] Pablo Arrighi's EITQ course, Pierre Wolpert's Introduction à la calculabilité, Arora&Barak's Computational Complexity: A Modern Approach

The Big Picture



Recap

- Turing Machines: automata with read/write memory

Recap

- Turing Machines: automata with read/write memory
- Church-Turing thesis: "Turing machines compute exactly functions that can be computed by any effective procedure"

Recap

- Turing Machines: automata with read/write memory
- Church-Turing thesis: "Turing machines compute exactly functions that can be computed by any effective procedure"
- Undecidable problems, e.g. the halting problem

Recap

- Turing Machines: automata with read/write memory
- Church-Turing thesis: "Turing machines compute exactly functions that can be computed by any effective procedure"
- Undecidable problems, e.g. the halting problem
- Nondeterminism does not affect computability

Recap

- Turing Machines: automata with read/write memory
- Church-Turing thesis: "Turing machines compute exactly functions that can be computed by any effective procedure"
- Undecidable problems, e.g. the halting problem
- Nondeterminism does not affect computability
- Complexity: efficiency

Recap

- Turing Machines: automata with read/write memory
- Church-Turing thesis: "Turing machines compute exactly functions that can be computed by any effective procedure"
- Undecidable problems, e.g. the halting problem
- Nondeterminism does not affect computability
- Complexity: efficiency
- Classes **P**, **NP**, ... to abstract away from hardware

Recap

- Turing Machines: automata with read/write memory
- Church-Turing thesis: "Turing machines compute exactly functions that can be computed by any effective procedure"
- Undecidable problems, e.g. the halting problem
- Nondeterminism does not affect computability
- Complexity: efficiency
- Classes **P**, **NP**, ... to abstract away from hardware
- Comparisons done with polynomial reductions $\leqslant_p$

The Travelling Salesman Problem (TS)

- Setup:
  - Set of n cities C
  - Distances between each pair $d(c_i, c_j) \in \mathbb{N}^*$
  - Some constant (threshold) t

The Travelling Salesman Problem (TS)

- Setup:
  - Set of n cities C
  - Distances between each pair $d(c_i, c_j) \in \mathbb{N}^*$
  - Some constant (threshold) t
- Question:

Is there a permutation σ of the cities such that

$$\sum_{i=1}^{n} d(c_{\sigma(i)}, c_{\sigma(i+1)}) + d(c_{\sigma(n)}, c_{\sigma(1)}) \leq t \ ?$$

I.e. is there an itinerary (a cycle) such that the total distance is smaller than t?

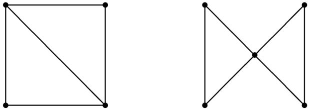
The Hamiltonian Cycle Problem (HC)

- Setup:
  - Graph $G = (V, E)$

The Hamiltonian Cycle Problem (HC)

- Setup:
  - Graph $G = (V, E)$
- Question:

  Is there a cycle in G that contains all vertices exactly once?



Solving HC by solving TS

HC $\leqslant_p$ TS

- Turn each vertex into a city $C = V$
- Set distance 1 if edge exists: $d(c_i, c_j) = 1$ if $(v_i, v_j) \in E$
- Set distance 2 otherwise: $d(c_i, c_j) = 2$ if $(v_i, v_j) \notin E$
- Set threshold $t = |V|$

Harder Classes

Definition

For any class of problems **C**, a problem $\mathcal{P}$ is **C-hard** if: $\forall \mathcal{P}_{\mathbf{C}},\ \mathcal{P}_{\mathbf{C}} \preccurlyeq_p \mathcal{P}$.
A problem $\mathcal{P}$ is **C-complete** if $\mathcal{P} \in \mathbf{C} \cap$ **C-hard**.

- **NP**: Problems for which <u>checking if an input is a solution</u> is efficient (in **P**)

Harder Classes

Definition

For any class of problems **C**, a problem $\mathcal{P}$ is **C-hard** if: $\forall \mathcal{P}_{\mathbf{C}},\ \mathcal{P}_{\mathbf{C}} \preccurlyeq_p \mathcal{P}$.
A problem $\mathcal{P}$ is **C-complete** if $\mathcal{P} \in \mathbf{C} \cap \mathbf{C}\text{-}\mathbf{hard}$.

- **NP**: Problems for which checking if an input is a solution is efficient (in **P**)
- Polynomial time nondeterministic Turing machine

Harder Classes

Definition

For any class of problems **C**, a problem $\mathcal{P}$ is **C-hard** if: $\forall \mathcal{P}_{\mathbf{C}},\ \mathcal{P}_{\mathbf{C}} \preccurlyeq_p \mathcal{P}$.
A problem $\mathcal{P}$ is **C-complete** if $\mathcal{P} \in \mathbf{C} \cap \mathbf{C}\text{-hard}$.

- **NP**: Problems for which <u>checking if an input is a solution</u> is efficient (in **P**)
- Polynomial time <u>nondeterministic</u> Turing machine
- Any **NP** problem reduces to any **NP-hard** problem

Harder Classes

Definition

For any class of problems **C**, a problem $\mathcal{P}$ is **C-hard** if: $\forall \mathcal{P}_{\mathbf{C}},\ \mathcal{P}_{\mathbf{C}} \preccurlyeq_p \mathcal{P}$.
A problem $\mathcal{P}$ is **C-complete** if $\mathcal{P} \in \mathbf{C} \cap \mathbf{C\text{-hard}}$.

- **NP**: Problems for which <u>checking if an input is a solution</u> is efficient (in **P**)
- Polynomial time <u>nondeterministic</u> Turing machine
- Any **NP** problem reduces to any **NP-hard** problem
- **NP-complete** problems are very well studied

NPC in Not Empty

SAT Problem: satisfiability of propositional calculus formulas in conjunctive normal form (CNF).

$$e.g.\ (p_2 \vee \neg p_1) \wedge (\neg p_3 \vee \neg p_2 \vee \neg p_4 \vee p_5)$$

NPC in Not Empty

SAT Problem: satisfiability of propositional calculus formulas in conjunctive normal form (CNF).

$$e.g.\ (p_2 \vee \neg p_1) \wedge (\neg p_3 \vee \neg p_2 \vee \neg p_4 \vee p_5)$$

Theorem

*SAT is **NP-complete**.*

- SAT is trivially in **NP**
- Any problem in **NP** is recognised by some nondeterministic polynomial Turing machine M. It is possible (and technical) to build a propositional formula B, in CNF, of polynomial size, that simulates M:
  1. it can be modified with little change to B_w given an input word w to the machine M
  2. B_w is satisfiable iff M accepts w, and a solution of B_w gives a solution of M on w

Brute Forcing

Theorem

Consider $L \in \boldsymbol{NP}$. There exists a deterministic Turing machine M such that M decides L and has a time complexity in $2^{n^{\mathcal{O}(1)}}$, i.e. $2^{p(n)}$ for some polynomial p.

Brute Forcing

Theorem

Consider $L \in \textbf{NP}$. There exists a deterministic Turing machine M such that M decides L and has a time complexity in $2^{n^{\mathcal{O}(1)}}$, i.e. $2^{p(n)}$ for some polynomial p.

Let M_{nd} be a nondeterministic machine of polynomial complexity $q(n)$ that accepts L. The idea is to simulate all executions of M_{nd} of length less than $q(n)$. For a word w, the machine M must thus:

1. Determine the length n of w and compute $q(n)$
2. Simulate each execution of M_{nd} of length $q(n)$ (let the time needed be $q'(n)$). If r is the largest number of possible choices within an execution of M_{nd}, there are at most $r^{q(n)}$ executions of length $q(n)$
3. If one of the simulated executions accepts, M accepts. Otherwise, M stops and rejects the word w

Complexity : bounded by $r^{q(n)}q'(n) = 2^{\log_2(r)q(n)+\log_2(q'(n))} \leq 2^{p(n)}$ for some p.

Exponential time hypothesis

- Most common "belief" is that $\mathbf{P} \neq \mathbf{NP}$

Exponential time hypothesis

- Most common "belief" is that $\mathbf{P} \neq \mathbf{NP}$
- Even stronger assumption: any algo solving SAT has complexity at least $2^{o(n)}$

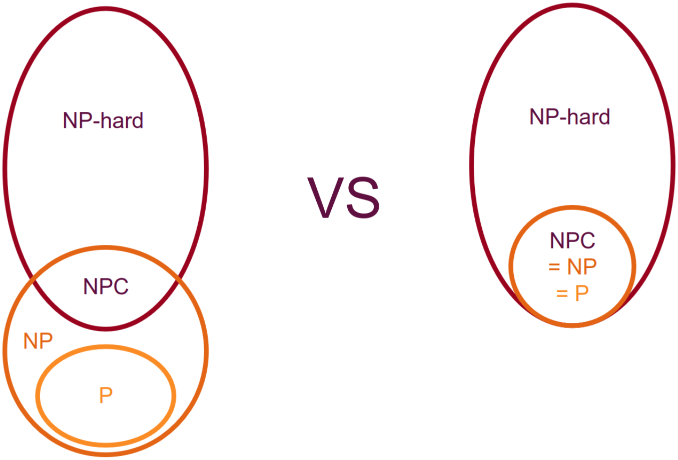
Exponential time hypothesis

- Most common "belief" is that $\mathbf{P} \neq \mathbf{NP}$
- Even stronger assumption: any algo solving SAT has complexity at least $2^{o(n)}$
- So far the best that can be done

Exponential time hypothesis

- Most common "belief" is that $\mathbf{P} \neq \mathbf{NP}$
- Even stronger assumption: any algo solving SAT has complexity at least $2^{o(n)}$
- So far the best that can be done
- Used as assumption for several results

The Big Picture



Proving **NP-completeness**

Lemma

NPC *is a polynomial equivalence class.*

Let $L_1, L_2 \in \textbf{NPC}$. Since $L_2 \in \textbf{NPC}$ and $L_1 \in \textbf{NP}$, $L_1 \leqslant_p L_2$. Similarly, $L_2 \leqslant_p L_1$, hence $L_1 \equiv_p L_2$.

Proving **NP-completeness**

Lemma

NPC *is a polynomial equivalence class.*

Let $L_1, L_2 \in \mathbf{NPC}$. Since $L_2 \in \mathbf{NPC}$ and $L_1 \in \mathbf{NP}$, $L_1 \preccurlyeq_p L_2$. Similarly, $L_2 \preccurlyeq_p L_1$, hence $L_1 \equiv_p L_2$.

How to prove that $L \in \mathbf{NPC}$?

- Do the same as for SAT

  or:
- Show that $L \in \mathbf{NP}$
- Show that another **NPC** problem polynomially reduces to L

Proving **NP-completeness**

Lemma

NPC *is a polynomial equivalence class.*

Let $L_1, L_2 \in \mathbf{NPC}$. Since $L_2 \in \mathbf{NPC}$ and $L_1 \in \mathbf{NP}$, $L_1 \leqslant_p L_2$. Similarly, $L_2 \leqslant_p L_1$, hence $L_1 \equiv_p L_2$.

How to prove that $L \in \mathbf{NPC}$?

- Do the same as for SAT
  or:
- Show that $L \in \mathbf{NP}$
- Show that another **NPC** problem polynomially reduces to L

Similarly, to show **NP-hardness** of problem L, it suffices to show that one **NPC** problem polynomially reduces to L.

Example: 3-SAT

3-SAT Problem: satisfiability of propositional calculus formulas in conjunctive normal form (CNF) with exactly 3 literals per clause.

Theorem

3-SAT $\in$ ***NP*** *and SAT* $\leqslant_p$ *3-SAT hence 3-SAT* $\in$ ***NPC***.

Example: 3-SAT

3-SAT Problem: satisfiability of propositional calculus formulas in conjunctive normal form (CNF) with exactly 3 literals per clause.

Theorem

3-SAT $\in$ ***NP*** *and SAT* $\leqslant_p$ *3-SAT hence 3-SAT* $\in$ ***NPC****.*

1. A clause $(x_1 \vee x_2)$ with two literals is replaced by
$$(x_1 \vee x_2 \vee y) \wedge (x_1 \vee x_2 \vee \neg y)$$
2. A clause (x_1) with a single literal is replaced by
$$(x_1 \vee y_1 \vee y_2) \wedge (x_1 \vee y_1 \vee \neg y_2) \wedge (x_1 \vee \neg y_1 \vee y_2) \wedge (x_1 \vee \neg y_1 \vee \neg y_2)$$
3. A clause $(x_1 \vee x_2 \vee \ldots \vee x_{\ell-1} \vee x_\ell)$ with $\ell \geq 4$ literals is replaced by
$$(x_1 \vee x_2 \vee y_2) \wedge (\neg y_2 \vee x_3 \vee y_3) \wedge (\neg y_3 \vee x_4 \vee y_4) \wedge \ldots$$
$$\wedge (\neg y_{\ell-3} \vee x_{\ell-2} \vee y_{\ell-2}) \wedge (\neg y_{\ell-2} \vee x_{\ell-1} \vee x_\ell)$$

Non-example: 2-SAT $\in$ **P**

Is all Hope Lost if $\mathbf{P} \neq \mathbf{NP}$?

- ⚠ Quantum computers may not a priori solve **NPC** problems efficiently!

Is all Hope Lost if $\mathbf{P} \neq \mathbf{NP}$?

- ⚠ Quantum computers may not a priori solve **NPC** problems efficiently!
- Exponential is the worst case. Algorithms efficient "on average" are possible

Is all Hope Lost if $\mathbf{P} \neq \mathbf{NP}$?

- ⚠ Quantum computers may not a priori solve **NPC** problems efficiently!
- Exponential is the worst case. Algorithms efficient "on average" are possible
- Some **P** problems are intractable (e.g. $\mathcal{O}(n^{12})$)

Is all Hope Lost if **P** $\neq$ **NP**?

- ⚠ Quantum computers may not a priori solve **NPC** problems efficiently!
- Exponential is the worst case. Algorithms efficient "on average" are possible
- Some **P** problems are intractable (e.g. $\mathcal{O}(n^{12})$)
- Heuristic methods to limit the exponential number of cases that need to be examined. $\rightarrow$ DPLL algorithm / SAT solvers

Is all Hope Lost if **P** $\neq$ **NP**?

- ⚠ Quantum computers may not a priori solve **NPC** problems efficiently!
- Exponential is the worst case. Algorithms efficient "on average" are possible
- Some **P** problems are intractable (e.g. $\mathcal{O}(n^{12})$)
- Heuristic methods to limit the exponential number of cases that need to be examined. $\rightarrow$ DPLL algorithm / SAT solvers
- Approximate solutions for optimisation problems

Is all Hope Lost if **P** $\neq$ **NP**?

- ⚠ Quantum computers may not a priori solve **NPC** problems efficiently!
- Exponential is the worst case. Algorithms efficient "on average" are possible
- Some **P** problems are intractable (e.g. $\mathcal{O}(n^{12})$)
- Heuristic methods to limit the exponential number of cases that need to be examined. $\rightarrow$ DPLL algorithm / SAT solvers
- Approximate solutions for optimisation problems
- Real-life problems have additional structure that can bring down to polynomial

Is all Hope Lost if **P** $\neq$ **NP**?

- ⚠ Quantum computers may not a priori solve **NPC** problems efficiently!
- Exponential is the worst case. Algorithms efficient "on average" are possible
- Some **P** problems are intractable (e.g. $\mathcal{O}(n^{12})$)
- Heuristic methods to limit the exponential number of cases that need to be examined. $\rightarrow$ DPLL algorithm / SAT solvers
- Approximate solutions for optimisation problems
- Real-life problems have additional structure that can bring down to polynomial
- Subexponential solutions possible. E.g. the Independent Set problem (IS) on planar graphs, in $O(2^{\sqrt{n}})$

Other Complexity Classes

- **coNP**: class of languages whose complement $(\Sigma^* \setminus L)$ is in **NP**

Other Complexity Classes

- **coNP**: class of languages whose complement ($\Sigma^* \setminus L$) is in **NP**
  - E.g. Is the number of solutions to a SAT formula 0?

Other Complexity Classes

- **coNP**: class of languages whose complement $(\Sigma^* \setminus L)$ is in **NP**
  - E.g. Is the number of solutions to a SAT formula 0?
- **EXPTIME**: class of languages decided by a deterministic Turing machine whose complexity function is bounded by $2^{n^{\mathcal{O}(1)}}$

Other Complexity Classes

- **coNP**: class of languages whose complement $(\Sigma^* \setminus L)$ is in **NP**
  - E.g. Is the number of solutions to a SAT formula 0?
- **EXPTIME**: class of languages decided by a deterministic Turing machine whose complexity function is bounded by $2^{n^{\mathcal{O}(1)}}$
- **PSPACE**: class of languages decided by a deterministic Turing machine whose space complexity (the number of tape cells used) is bounded by a polynomial

Other Complexity Classes

- **coNP**: class of languages whose complement $(\Sigma^* \setminus L)$ is in **NP**
  - E.g. Is the number of solutions to a SAT formula 0?
- **EXPTIME**: class of languages decided by a deterministic Turing machine whose complexity function is bounded by $2^{n^{\mathcal{O}(1)}}$
- **PSPACE**: class of languages decided by a deterministic Turing machine whose space complexity (the number of tape cells used) is bounded by a polynomial
- **NPSPACE**: class of languages accepted by a **non**deterministic Turing machine whose space complexity is bounded by a polynomial

Other Complexity Classes

- **coNP**: class of languages whose complement $(\Sigma^* \setminus L)$ is in **NP**
  - E.g. Is the number of solutions to a SAT formula 0?
- **EXPTIME**: class of languages decided by a deterministic Turing machine whose complexity function is bounded by $2^{n^{\mathcal{O}(1)}}$
- **PSPACE**: class of languages decided by a deterministic Turing machine whose space complexity (the number of tape cells used) is bounded by a polynomial
- **NPSPACE**: class of languages accepted by a **non**deterministic Turing machine whose space complexity is bounded by a polynomial
- **EXPSPACE**: class of languages decided by a deterministic Turing machine whose space complexity is bounded by $2^{n^{\mathcal{O}(1)}}$

Other Complexity Classes

- **coNP**: class of languages whose complement $(\Sigma^* \setminus L)$ is in **NP**
  - E.g. Is the number of solutions to a SAT formula 0?
- **EXPTIME**: class of languages decided by a deterministic Turing machine whose complexity function is bounded by $2^{n^{\mathcal{O}(1)}}$
- **PSPACE**: class of languages decided by a deterministic Turing machine whose space complexity (the number of tape cells used) is bounded by a polynomial
- **NPSPACE**: class of languages accepted by a **non**deterministic Turing machine whose space complexity is bounded by a polynomial
- **EXPSPACE**: class of languages decided by a deterministic Turing machine whose space complexity is bounded by $2^{n^{\mathcal{O}(1)}}$

$$\mathbf{P} \subseteq \begin{array}{c}\mathbf{NP}\\ \mathbf{coNP}\end{array} \subseteq \begin{array}{c}\mathbf{PSPACE}\\ =\\ \mathbf{NPSPACE}\end{array} \subseteq \mathbf{EXPTIME} \subseteq \mathbf{EXPSPACE}$$

NP-intermediate (**NPI**)

$$\mathbf{NPI} := \mathbf{NP} \setminus (\mathbf{P} \cup \mathbf{NPC})$$

NP-intermediate (**NPI**)

$$\mathbf{NPI} := \mathbf{NP} \setminus (\mathbf{P} \cup \mathbf{NPC})$$

Theorem

$$\boldsymbol{P} \neq \boldsymbol{NP} \iff \boldsymbol{NPI} \neq \emptyset$$

NP-intermediate (**NPI**)

$$\mathbf{NPI} := \mathbf{NP} \setminus (\mathbf{P} \cup \mathbf{NPC})$$

Theorem

$$\boldsymbol{P} \neq \boldsymbol{NP} \iff \boldsymbol{NPI} \neq \emptyset$$

Some candidates for **NPI** problems:

- Factoring integers: Does n have a factor in $[2, k]$?
- Graph isomorphism problem: Given two graphs, is there a bijection between the sets of vertices that preserve connections?
- Triviality of a knot

The Polynomial Hierarchy (1/2)

Definition

Let $L \subseteq \Sigma^*$ and p be a polynomial. We define:

$$\exists^p L := \{x \in \Sigma^* \mid \exists w \in \Sigma^*,\ |w| \leq p(|x|) \implies \langle x, w \rangle \in L\}$$
$$\forall^p L := \{x \in \Sigma^* \mid \forall w \in \Sigma^*,\ |w| \leq p(|x|) \implies \langle x, w \rangle \in L\}$$

The Polynomial Hierarchy (1/2)

Definition

Let $L \subseteq \Sigma^*$ and p be a polynomial. We define:

$$\exists^p L := \{x \in \Sigma^* \mid \exists w \in \Sigma^*,\ |w| \leq p(|x|) \implies \langle x, w \rangle \in L\}$$
$$\forall^p L := \{x \in \Sigma^* \mid \forall w \in \Sigma^*,\ |w| \leq p(|x|) \implies \langle x, w \rangle \in L\}$$

Let $\mathbf{C}$ be class of languages. We define:

$$\exists^P \mathbf{C} := \{\exists^p L \mid p \text{ is a polynomial}, L \in \mathbf{C}\}$$
$$\forall^P \mathbf{C} := \{\forall^p L \mid p \text{ is a polynomial}, L \in \mathbf{C}\}$$
$$\mathbf{coC} := \{\Sigma^* \setminus L \mid L \in \mathbf{C}\}$$

The Polynomial Hierarchy (1/2)

Definition

Let $L \subseteq \Sigma^*$ and p be a polynomial. We define:

$$\exists^p L := \{x \in \Sigma^* \mid \exists w \in \Sigma^*,\ |w| \leq p(|x|) \implies \langle x, w\rangle \in L\}$$
$$\forall^p L := \{x \in \Sigma^* \mid \forall w \in \Sigma^*,\ |w| \leq p(|x|) \implies \langle x, w\rangle \in L\}$$

Let $\mathbf{C}$ be class of languages. We define:

$$\exists^P \mathbf{C} := \{\exists^p L \mid p \text{ is a polynomial}, L \in \mathbf{C}\}$$
$$\forall^P \mathbf{C} := \{\forall^p L \mid p \text{ is a polynomial}, L \in \mathbf{C}\}$$
$$\mathbf{coC} := \{\Sigma^* \setminus L \mid L \in \mathbf{C}\}$$

- $\mathbf{co}\exists^P\mathbf{C} = \forall^P\mathbf{coC}$ and $\mathbf{co}\forall^P\mathbf{C} = \exists^P\mathbf{coC}$
- $\mathbf{NP} = \exists^P\mathbf{P}$ and $\mathbf{coNP} = \forall^P\mathbf{P}$
- $\exists^q\exists^p L = \{x{\in}\Sigma^* | \exists w_1, w_2 {\in} \Sigma^*,\ |w_1| {\leq} p(|x|),\ |w_2| {\leq} q(|x|),\ \langle x, w_1, w_2\rangle {\in} L\} = \exists^{p+q} L$ hence $\exists^P\exists^P\mathbf{C} = \exists^P\mathbf{C}$.

The Polynomial Hierarchy (2/2)

Definition

We define the following classes:

$$
\begin{aligned}
\Sigma_0^P = \Pi_0^P &:= \mathbf{P} \\
\Sigma_{k+1}^P &:= \exists^P \Pi_k^P \\
\Pi_{k+1}^P &:= \forall^P \Sigma_k^P \\
\mathbf{PH} &:= \bigcup_{k \in \mathbb{N}} \Sigma_k^P = \bigcup_{k \in \mathbb{N}} \Pi_k^P
\end{aligned}
$$

The Polynomial Hierarchy (2/2)

Definition

We define the following classes:

$$
\begin{aligned}
\Sigma_0^P = \Pi_0^P &:= \mathbf{P} \\
\Sigma_{k+1}^P &:= \exists^P \Pi_k^P \\
\Pi_{k+1}^P &:= \forall^P \Sigma_k^P \\
\mathbf{PH} &:= \bigcup_{k \in \mathbb{N}} \Sigma_k^P = \bigcup_{k \in \mathbb{N}} \Pi_k^P
\end{aligned}
$$

$$
\begin{tikzcd}
\vdots & & \vdots \\
\Sigma_2^P & & \Pi_2^P \\
\mathbf{NP} = \Sigma_1^P \arrow[u] \arrow[urr] & & \Pi_1^P = \mathbf{coNP} \arrow[u] \arrow[ull] \\
& \Sigma_0^P = \mathbf{P} = \Pi_0^P \arrow[ul] \arrow[ur] &
\end{tikzcd}
$$

Inclusion graph

The Polynomial Hierarchy (2/2)

Definition

We define the following classes:

$$
\begin{aligned}
\Sigma_0^P = \Pi_0^P &:= \mathbf{P} \\
\Sigma_{k+1}^P &:= \exists^P \Pi_k^P \\
\Pi_{k+1}^P &:= \forall^P \Sigma_k^P \\
\mathbf{PH} &:= \bigcup_{k \in \mathbb{N}} \Sigma_k^P = \bigcup_{k \in \mathbb{N}} \Pi_k^P
\end{aligned}
$$

$$
\begin{tikzcd}
\vdots & & \vdots \\
\Sigma_2^P & & \Pi_2^P \\
\mathbf{NP} = \Sigma_1^P \arrow[u] \arrow[urr] & & \Pi_1^P = \mathbf{coNP} \arrow[u] \arrow[ull] \\
& \Sigma_0^P = \mathbf{P} = \Pi_0^P \arrow[ul] \arrow[ur] &
\end{tikzcd}
$$

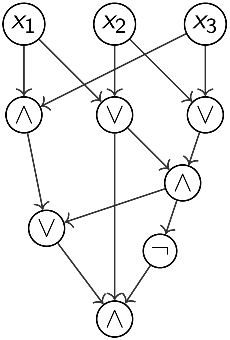
Inclusion graph

- $\mathbf{P} = \mathbf{NP} \iff \mathbf{P} = \mathbf{PH}$
  - "The polynomial hierarchy collapses to **P**"
- $\mathbf{NP} = \mathbf{coNP} \implies \mathbf{NP} = \mathbf{PH}$
  - "The polynomial hierarchy collapses to its second level"

Example: Circuit Minimisation (CM) (1/3)

Beyond **NP** and **coNP**

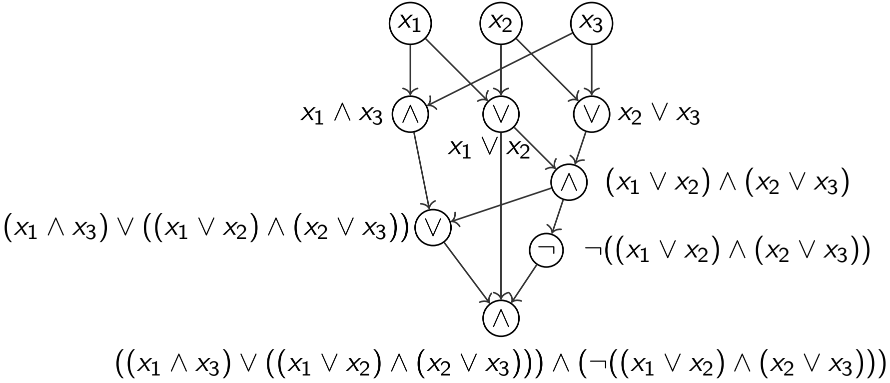
Boolean circuit: Labelled directed acyclic graph, with a unique "leaf" (or "sink"). "Coleaves" are labelled by (distinct) boolean variables $\{x_1, x_2, ...\}$ and the other vertices are labelled by elements of $\{\neg, \vee, \wedge\}$. E.g.:



Example: Circuit Minimisation (CM) (1/3)

Beyond **NP** and **coNP**

Boolean circuit: Labelled directed acyclic graph, with a unique "leaf" (or "sink"). "Coleaves" are labelled by (distinct) boolean variables $\{x_1, x_2, \ldots\}$ and the other vertices are labelled by elements of $\{\neg, \vee, \wedge\}$. E.g.:



Example: Circuit Minimisation (CM) (2/3)

Beyond **NP** and **coNP**

- Setup:
  - Boolean circuit A
  - Some constant (threshold) t

Example: Circuit Minimisation (CM) (2/3)

- Setup:
  - Boolean circuit A
  - Some constant (threshold) t
- Question:

  Is there a boolean circuit B with at most t gates that computes the same boolean function as A?

Example: Circuit Minimisation (CM) (3/3)

Beyond **NP** and **coNP**

Claim: CM $\in \Sigma_2^P$.

Intuitively, it is not enough to be given a solution B to check, one still has to check that $A(x) = B(x)$ for all x.

Example: Circuit Minimisation (CM) (3/3)

Beyond **NP** and **coNP**

Claim: CM $\in \Sigma_2^P$.

Intuitively, it is not enough to be given a solution B to check, one still has to check that $A(x) = B(x)$ for all x.

Formally:

$$L = \{\langle A, t, B, x \rangle \mid B \text{ has at most } t \text{ gates, and } A(x) = B(x)\} \in \mathbf{P}$$

and

$$\mathrm{CM} = \exists^t \forall^{|\operatorname{in}(A)|} L \in \Sigma_2^P$$

Probabilistic Turing Machines

- Intuition: In a nondeterministic Turing machine, maybe there are more accepting paths than rejecting ones (for accepted words w). Maybe there are ways to choose paths that have a better chance of acceptance.

Probabilistic Turing Machines

- Intuition: In a nondeterministic Turing machine, maybe there are more accepting paths than rejecting ones (for accepted words w). Maybe there are ways to choose paths that have a better chance of acceptance.

- Deterministic Turing machines are a subset of probabilistic Turing machines

Probabilistic Turing Machines

- Intuition: In a nondeterministic Turing machine, maybe there are more accepting paths than rejecting ones (for accepted words w). Maybe there are ways to choose paths that have a better chance of acceptance.
- Deterministic Turing machines are a subset of probabilistic Turing machines
- Probabilistic Turing machine (PTM): $\mathcal{M} = (Q, \Sigma, \_, \Gamma, \delta_1, \delta_2, s, F)$
  - $\delta_1, \delta_2 : Q \times \Gamma \rightharpoonup Q \times \Gamma \times \{\triangleright, \square, \triangleleft\}$, the two transition functions, picked uniformly at random at each step

Probabilistic Turing Machines

- Intuition: In a nondeterministic Turing machine, maybe there are more accepting paths than rejecting ones (for accepted words w). Maybe there are ways to choose paths that have a better chance of acceptance.

- Deterministic Turing machines are a subset of probabilistic Turing machines

- Probabilistic Turing machine (PTM): $\mathcal{M} = (Q, \Sigma, \_, \Gamma, \delta_1, \delta_2, s, F)$
  - $\delta_1, \delta_2 : Q \times \Gamma \rightharpoonup Q \times \Gamma \times \{\triangleright, \square, \triangleleft\}$, the two transition functions, picked uniformly at random at each step

  $\mathcal{M}$ accepts L with error probability ϵ if:
  - $w \in L \implies \Pr[\mathcal{M} \text{ accepts } w] \geq 1 - \epsilon$
  - $w \notin L \implies \Pr[\mathcal{M} \text{ rejects } w] \geq 1 - \epsilon$

Probabilistic Complexity Class **PP**

Definition (Probabilistic Polynomial (**PP**))

Language $L \in \mathbf{PP}$ if there exists polynomial-time PTM such that:

- $w \in L \implies \Pr[\mathcal{M} \text{ accepts } w] > \frac{1}{2}$
- $w \notin L \implies \Pr[\mathcal{M} \text{ accepts } w] < \frac{1}{2}$

Probabilistic Complexity Class **PP**

Definition (Probabilistic Polynomial (**PP**))

Language $L \in \mathbf{PP}$ if there exists polynomial-time PTM such that:

- $w \in L \implies \Pr[\mathcal{M} \text{ accepts } w] > \frac{1}{2}$
- $w \notin L \implies \Pr[\mathcal{M} \text{ accepts } w] < \frac{1}{2}$

Obviously $\mathbf{P} \subseteq \mathbf{PP}$.

Probabilistic Complexity Class **PP**

Definition (Probabilistic Polynomial (**PP**))

Language $L \in \mathbf{PP}$ if there exists polynomial-time PTM such that:

- $w \in L \implies \Pr[\mathcal{M} \text{ accepts } w] > \frac{1}{2}$
- $w \notin L \implies \Pr[\mathcal{M} \text{ accepts } w] < \frac{1}{2}$

Obviously $\mathbf{P} \subseteq \mathbf{PP}$.

Seems reasonable, but:

Theorem

$\boldsymbol{NP} \subseteq \boldsymbol{PP}$

Proof: show that SAT $\in \mathbf{PP}$

SAT $\in$ **PP**

Consider SAT formula $F(p_1, \ldots, p_n)$ in n variables $p_1, \ldots, p_n$. We build a probabilistic algorithm:

SAT $\in$ **PP**

Consider SAT formula $F(p_1, ..., p_n)$ in n variables $p_1, ..., p_n$. We build a probabilistic algorithm:

- Choose a value x_i between True and False uniformly at random, for each p_i

SAT $\in$ **PP**

Consider SAT formula $F(p_1, ..., p_n)$ in n variables $p_1, ..., p_n$. We build a probabilistic algorithm:

- Choose a value x_i between True and False uniformly at random, for each p_i
- Check if $F(x_1, ..., x_n)$ is True
  - If True: accept
  - If False: accept with proba $\frac{1}{2} - \frac{1}{2^{n+1}}$ and reject with proba $\frac{1}{2} + \frac{1}{2^{n+1}}$

SAT $\in$ **PP**

Consider SAT formula $F(p_1, ..., p_n)$ in n variables $p_1, ..., p_n$. We build a probabilistic algorithm:

- Choose a value x_i between True and False uniformly at random, for each p_i
- Check if $F(x_1, ..., x_n)$ is True
  - If True: accept
  - If False: accept with proba $\frac{1}{2} - \frac{1}{2^{n+1}}$ and reject with proba $\frac{1}{2} + \frac{1}{2^{n+1}}$

Let $\mathcal{M}$ be a PTM that implements it, and let x be the number of satisfying assignments to F (SAT asks whether $x = 0$ or not). Then:

$$\Pr[\mathcal{M} \text{ accepts } F] = \frac{x}{2^n} \times 1 + \left(1 - \frac{x}{2^n}\right) \times \left(\frac{1}{2} - \frac{1}{2^{n+1}}\right) = \frac{1}{2} + \frac{1}{2^{n+1}}(2x - 1)$$

SAT $\in$ **PP**

Consider SAT formula $F(p_1, \ldots, p_n)$ in n variables $p_1, \ldots, p_n$. We build a probabilistic algorithm:

- Choose a value x_i between True and False uniformly at random, for each p_i
- Check if $F(x_1, \ldots, x_n)$ is True
  - If True: accept
  - If False: accept with proba $\frac{1}{2} - \frac{1}{2^{n+1}}$ and reject with proba $\frac{1}{2} + \frac{1}{2^{n+1}}$

Let $\mathcal{M}$ be a PTM that implements it, and let x be the number of satisfying assignments to F (SAT asks whether $x = 0$ or not). Then:

$$\Pr[\mathcal{M} \text{ accepts } F] = \frac{x}{2^n} \times 1 + \left(1 - \frac{x}{2^n}\right) \times \left(\frac{1}{2} - \frac{1}{2^{n+1}}\right) = \frac{1}{2} + \frac{1}{2^{n+1}}(2x - 1)$$

- If $x > 0$ (F satisfiable), $\Pr[\mathcal{M} \text{ accepts } F] > \frac{1}{2}$
- If $x = 0$ (F unsatisfiable), $\Pr[\mathcal{M} \text{ accepts } F] < \frac{1}{2}$

A **PP-complete** Problem

Definition (Majority Problem)

Let F be a SAT formula over $p_1, \ldots, p_n$, and $s = |\{\vec{x} \mid F(\vec{x}) = \mathit{True}\}|$, the number of solutions to F.
Decide whether $s < 2^{n-1}$ or $s \geq 2^{n-1}$.

A **PP-complete** Problem

Definition (Majority Problem)

Let F be a SAT formula over $p_1, \ldots, p_n$, and $s = |\{\vec{x} \mid F(\vec{x}) = \mathit{True}\}|$, the number of solutions to F.
Decide whether $s < 2^{n-1}$ or $s \geq 2^{n-1}$.

Theorem

Majority $\in$ ***PP-complete***

A **PP-complete** Problem

Definition (Majority Problem)

Let F be a SAT formula over $p_1, \ldots, p_n$, and $s = |\{\vec{x} \mid F(\vec{x}) = \mathit{True}\}|$, the number of solutions to F.
Decide whether $s < 2^{n-1}$ or $s \geq 2^{n-1}$.

Theorem

Majority $\in$ ***PP-complete***

This is very close to a counting problem. See next session.

Probabilistic Complexity Class **BPP**

Definition (Bounded Probabilistic Polynomial (**BPP**))

Language $L \in \mathbf{BPP}$ if there exists polynomial-time PTM such that:

- $w \in L \implies \Pr[\mathcal{M} \text{ accepts } w] > \frac{2}{3}$
- $w \notin L \implies \Pr[\mathcal{M} \text{ accepts } w] < \frac{1}{3}$

i.e. L is recognised by a PTM with error probability $\frac{1}{3}$.

Probabilistic Complexity Class BPP

Definition (Bounded Probabilistic Polynomial (BPP))

Language $L \in \mathbf{BPP}$ if there exists polynomial-time PTM such that:

- $w \in L \implies \Pr[\mathcal{M} \text{ accepts } w] > \frac{2}{3}$
- $w \notin L \implies \Pr[\mathcal{M} \text{ accepts } w] < \frac{1}{3}$

i.e. L is recognised by a PTM with error probability $\frac{1}{3}$.

Obviously $\mathbf{P} \subseteq \mathbf{BPP} \subseteq \mathbf{PP}$.

Probabilistic Complexity Class **BPP**

Definition (Bounded Probabilistic Polynomial (**BPP**))

Language $L \in \mathbf{BPP}$ if there exists polynomial-time PTM such that:

- $w \in L \implies \Pr[\mathcal{M} \text{ accepts } w] > \frac{2}{3}$
- $w \notin L \implies \Pr[\mathcal{M} \text{ accepts } w] < \frac{1}{3}$

i.e. L is recognised by a PTM with error probability $\frac{1}{3}$.

Obviously $\mathbf{P} \subseteq \mathbf{BPP} \subseteq \mathbf{PP}$.

Lemma

For any polynomial-time PTM $\mathcal{M}_1$ with error probability $\frac{1}{3}$, there exists equivalent polynomial-time PTM $\mathcal{M}_2$ with error probability $\frac{1}{2^n}$

Probabilistic Complexity Class **BPP**

Definition (Bounded Probabilistic Polynomial (**BPP**))

Language $L \in$ **BPP** if there exists polynomial-time PTM such that:

- $w \in L \implies \Pr[\mathcal{M} \text{ accepts } w] > \frac{2}{3}$
- $w \notin L \implies \Pr[\mathcal{M} \text{ accepts } w] < \frac{1}{3}$

i.e. L is recognised by a PTM with error probability $\frac{1}{3}$.

Obviously $\mathbf{P} \subseteq \mathbf{BPP} \subseteq \mathbf{PP}$.

Lemma

For any polynomial-time PTM $\mathcal{M}_1$ with error probability $\frac{1}{3}$, there exists equivalent polynomial-time PTM $\mathcal{M}_2$ with error probability $\frac{1}{2^n}$

Example of BPP problem: equivalence of Read-Once Branching Programs (ROBPs).

A Probabilistic Version of **NP**?

- Interactive proof system: verifier (us with reasonable computational power) interacts with a prover (some oracle with infinite power)

A Probabilistic Version of **NP**?

- Interactive proof system: verifier (us with reasonable computational power) interacts with a prover (some oracle with infinite power)
  - If w accepted and prover is honest, verifier can be convinced
  - If w rejected, any prover can only convince with very small probability

A Probabilistic Version of **NP**?

- Interactive proof system: verifier (us with reasonable computational power) interacts with a prover (some oracle with infinite power)
  - If w accepted and prover is honest, verifier can be convinced
  - If w rejected, any prover can only convince with very small probability
- **NP**: verifier is a poly-time deterministic TM, with just one message from the prover (who has access to the input)

A Probabilistic Version of **NP**?

- Interactive proof system: verifier (us with reasonable computational power) interacts with a prover (some oracle with infinite power)
  - If w accepted and prover is honest, verifier can be convinced
  - If w rejected, any prover can only convince with very small probability
- **NP**: verifier is a poly-time deterministic TM, with just one message from the prover (who has access to the input)
- Arthur-Merlin protocol: verifier (poly-time PTM) is called Arthur, and the prover is called Merlin.

A Probabilistic Version of **NP**?

- Interactive proof system: verifier (us with reasonable computational power) interacts with a prover (some oracle with infinite power)
  - If w accepted and prover is honest, verifier can be convinced
  - If w rejected, any prover can only convince with very small probability
- **NP**: verifier is a poly-time deterministic TM, with just one message from the prover (who has access to the input)
- Arthur-Merlin protocol: verifier (poly-time PTM) is called Arthur, and the prover is called Merlin.
- **MA**: Languages recognised by an Arthur-Merlin protocol with a single message (from Merlin to Arthur), and with error probability $\frac{1}{3}$

A Probabilistic Version of **NP**?

- Interactive proof system: verifier (us with reasonable computational power) interacts with a prover (some oracle with infinite power)
  - If w accepted and prover is honest, verifier can be convinced
  - If w rejected, any prover can only convince with very small probability
- **NP**: verifier is a poly-time deterministic TM, with just one message from the prover (who has access to the input)
- Arthur-Merlin protocol: verifier (poly-time PTM) is called Arthur, and the prover is called Merlin.
- **MA**: Languages recognised by an Arthur-Merlin protocol with a single message (from Merlin to Arthur), and with error probability $\frac{1}{3}$
- **NP** $\subseteq$ **MA**, and **BPP** $\subseteq$ **MA**

A Probabilistic Version of **NP**?

- Interactive proof system: verifier (us with reasonable computational power) interacts with a prover (some oracle with infinite power)
  - If w accepted and prover is honest, verifier can be convinced
  - If w rejected, any prover can only convince with very small probability
- **NP**: verifier is a poly-time deterministic TM, with just one message from the prover (who has access to the input)
- Arthur-Merlin protocol: verifier (poly-time PTM) is called Arthur, and the prover is called Merlin.
- **MA**: Languages recognised by an Arthur-Merlin protocol with a single message (from Merlin to Arthur), and with error probability $\frac{1}{3}$
- $\mathbf{NP} \subseteq \mathbf{MA}$, and $\mathbf{BPP} \subseteq \mathbf{MA}$
- $\mathbf{P} = \mathbf{BPP} \implies \mathbf{NP} = \mathbf{MA}$

A Probabilistic Version of **NP**?

- Interactive proof system: verifier (us with reasonable computational power) interacts with a prover (some oracle with infinite power)
  - If w accepted and prover is honest, verifier can be convinced
  - If w rejected, any prover can only convince with very small probability
- **NP**: verifier is a poly-time deterministic TM, with just one message from the prover (who has access to the input)
- Arthur-Merlin protocol: verifier (poly-time PTM) is called Arthur, and the prover is called Merlin.
- **MA**: Languages recognised by an Arthur-Merlin protocol with a single message (from Merlin to Arthur), and with error probability $\frac{1}{3}$
- $\mathbf{NP} \subseteq \mathbf{MA}$, and $\mathbf{BPP} \subseteq \mathbf{MA}$
- $\mathbf{P} = \mathbf{BPP} \implies \mathbf{NP} = \mathbf{MA}$
- $\mathbf{MA} \subseteq \Sigma_2^P$, and $\mathbf{MA} \subseteq \Pi_2^P$

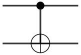
A Circuit-Based Definition of **P** and **NP**

Definition (Reversible Circuit)

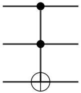
A reversible circuit is a boolean circuit composed of the logic gates:

$|0\rangle$ —

: $|x\rangle \mapsto |x \oplus 1\rangle$



: $|x, y\rangle \mapsto |x, y \oplus x\rangle$



: $|x, y, z\rangle \mapsto |x, y, z \oplus xy\rangle$

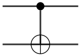
A Circuit-Based Definition of **P** and **NP**

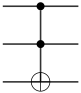
Definition (Reversible Circuit)

A reversible circuit is a boolean circuit composed of the logic gates:

$|0\rangle$

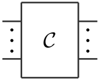
 : $|x\rangle \mapsto |x \oplus 1\rangle$

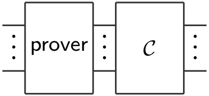
 : $|x, y\rangle \mapsto |x, y \oplus x\rangle$

 : $|x, y, z\rangle \mapsto |x, y, z \oplus xy\rangle$

$L \in$ **P**: | $L \in$ **NP**:

$\exists \mathcal{C}$ a poly-sized uniform family of reversible boolean circuits, such that in:





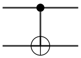
A Circuit-Based Definition of **P** and **NP**

Definition (Reversible Circuit)

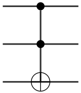
A reversible circuit is a boolean circuit composed of the logic gates:

$|0\rangle$ ——

$: |x\rangle \mapsto |x \oplus 1\rangle$



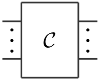
$: |x, y\rangle \mapsto |x, y \oplus x\rangle$

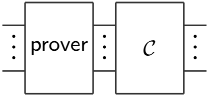


$: |x, y, z\rangle \mapsto |x, y, z \oplus xy\rangle$

$L \in \mathbf{P}$:	$L \in \mathbf{NP}$:

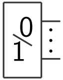
$\exists \mathcal{C}$ a poly-sized uniform family of reversible boolean circuits, such that in:



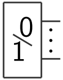


- If input $\in L$ and prover honest, first output is $|1\rangle$
- If input $\notin L$, first output is $|0\rangle$

A Circuit-Based Definition of **BPP** and **MA**

We denote  a (uniform) random bitstring generator.

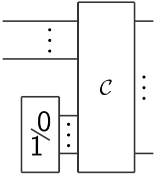
A Circuit-Based Definition of **BPP** and **MA**

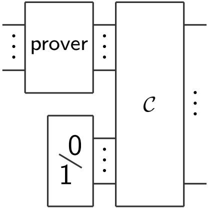
We denote 

a (uniform) random bitstring generator.

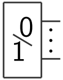
$L \in$ **BPP**:	$L \in$ **MA**:

$\exists\mathcal{C}$ a poly-sized uniform family of reversible boolean circuits, such that in:



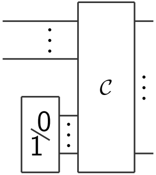


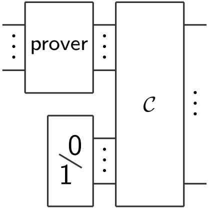
A Circuit-Based Definition of **BPP** and **MA**

We denote  a (uniform) random bitstring generator.

$L \in$ **BPP**:	$L \in$ **MA**:

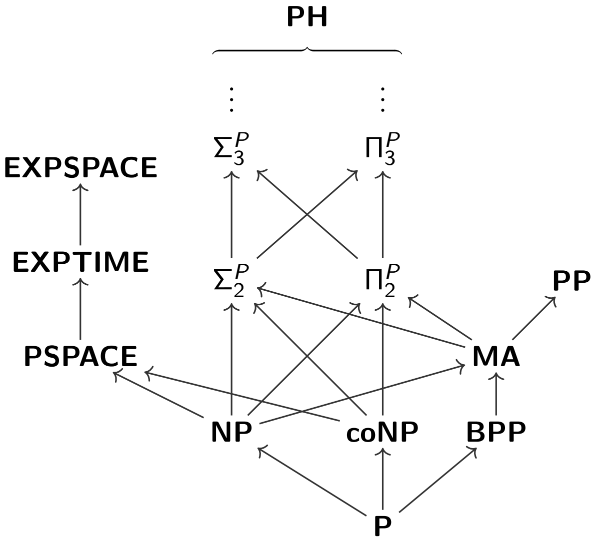
$\exists \mathcal{C}$ a poly-sized uniform family of reversible boolean circuits, such that in:





- If input $\in L$ and prover honest, $\Pr[\text{first output is } |1\rangle] \geq \frac{2}{3}$
- If input $\notin L$, $\Pr[\text{first output is } |1\rangle] \leq \frac{1}{3}$

Overview of Complexity Classes



Inclusion graph