Université Paris Cité
LIPADE

Algorithmic Complexity

Non-Deterministic Time

Jean-Guy Mailly (jean-guy.mailly@u-paris.fr)

2022

Outline

Non-Deterministic Time Complexity Classes

Polynomial Hierarchy

Complexity of Well-Known Problems
- SAT and Related Problems
- Other Theoretical Problems
- Video Games

Determining the Complexity of a Problem

Reminder on DTM vs NDTM [Turing 1936]

Deterministic

$$
\begin{tikzcd}
(q_0, x_0) \arrow[d, "\delta"'] \\
(q_1, x_1) \arrow[d, "\delta"'] \\
\cdots \arrow[d, "\delta"'] \\
(q_n, x_n)
\end{tikzcd}
$$

Non-Deterministic

$$
\begin{tikzcd}
 & & (q_0, x_0) \arrow[dl, "\delta"'] \arrow[d, "\delta"'] \arrow[dr, "\delta"] & \\
 & (q_1, x_1) \arrow[dl, "\delta"'] & \cdots \arrow[d, "\delta"'] & (q_k, x_k) \arrow[d, "\delta"'] \\
\cdots \arrow[d, "\delta"'] & & \cdots \arrow[dl, "\delta"'] \arrow[d, "\delta"'] & \cdots \arrow[d, "\delta"'] \\
(q_m, x_m) & (q_n, x_n) & (q_o, x_o) & (q_p, x_p)
\end{tikzcd}
$$

Intuition on Solving an Exponential Problem...

. . . ... with DTM

- Linear calculations since δ is a 1 to 1 mapping from configurations to transitions
- **Exponential number of steps** cannot be avoided

Intuition on Solving an Exponential Problem...

. . . ... with DTM

- Linear calculations since δ is a 1 to 1 mapping from configurations to transitions
- **Exponential number of steps** cannot be avoided

. . . ... with NDTM

- The tree structure can simulate parallel computing
- The solving time is the length of the longest branch of the tree
- **COULD BE polynomial** (no guarantee in general)
- When it stays exponential, it **COULD BE smaller exponential** (e.g. $\mathcal{O}(2^n)$ steps instead of $\mathcal{O}(10^n)$)

Really Naive Example

Solving an Equation...

Does $f(n) = 0$ have a solution, with $n \in [0, 1, \ldots, 10^9]$?

Really Naive Example

Solving an Equation...

Does $f(n) = 0$ have a solution, with $n \in [0, 1, \dots, 10^9]$?

. . . ... with DTM

- Compute $f(0)$. If it works, fine, otherwise compute $f(1)$, then $f(2)$,. . .

Really Naive Example

Solving an Equation...

Does $f(n) = 0$ have a solution, with $n \in [0, 1, \dots, 10^9]$?

...... with DTM

- Compute $f(0)$. If it works, fine, otherwise compute $f(1)$, then $f(2), \dots$
- Not efficient: if the solutions of the equation are huge (e.g. 10^9), then a lot of useless calculations are made

Really Naive Example

Solving an Equation...

Does $f(n) = 0$ have a solution, with $n \in [0, 1, \ldots, 10^9]$?

. . . ... with DTM

- Compute $f(0)$. If it works, fine, otherwise compute $f(1)$, then $f(2)$,. . .
- Not efficient: if the solutions of the equation are huge (e.g. 10^9), then a lot of useless calculations are made

. . . ... with NDTM

- Compute $f(i)$ on the i^{th} branch of the tree, with $0 \leq i \leq 10^9$

Really Naive Example

Solving an Equation...

Does $f(n) = 0$ have a solution, with $n \in [0, 1, \ldots, 10^9]$?

. . . ... with DTM

- Compute $f(0)$. If it works, fine, otherwise compute $f(1)$, then $f(2)$,. . .
- Not efficient: if the solutions of the equation are huge (e.g. 10^9), then a lot of useless calculations are made

. . . ... with NDTM

- Compute $f(i)$ on the i^{th} branch of the tree, with $0 \leq i \leq 10^9$
- Whatever the solution of the problem, it is obtained in the time of a « single » $f(i)$ computation

Non-Deterministic Complexity Classes

Evaluating Time with NDTM

Given a function $f : \mathbb{N} \mapsto \mathbb{N}$, NTIME($f(n)$) is the set of all languages decided by a NDTM $\mathcal{M}$ in less than $g(n)$ steps (longer branch), with $g(n) \in \mathcal{O}(f(n))$

Non-Deterministic Complexity Classes

Evaluating Time with NDTM

Given a function $f : \mathbb{N} \mapsto \mathbb{N}$, NTIME($f(n)$) is the set of all languages decided by a NDTM $\mathcal{M}$ in less than $g(n)$ steps (longer branch), with $g(n) \in \mathcal{O}(f(n))$

Proposition

- $\forall f : \mathbb{N} \mapsto \mathbb{N}$, then DTIME($f(n)$) $\subseteq$ NTIME($f(n)$)
- $\forall f(n) \geq n$, NTIME($f(n)$) is closed for finite union and finite intersection
  - if $\mathcal{L}_1, \ldots, \mathcal{L}_m \in$ NTIME($f(n)$), then $\mathcal{L}_1 \cup \cdots \cup \mathcal{L}_m \in$ NTIME($f(n)$)
  - if $\mathcal{L}_1, \ldots, \mathcal{L}_m \in$ NTIME($f(n)$), then $\mathcal{L}_1 \cap \cdots \cap \mathcal{L}_m \in$ NTIME($f(n)$)

Non-Deterministic Complexity Classes

Evaluating Time with NDTM

Given a function $f : \mathbb{N} \mapsto \mathbb{N}$, NTIME($f(n)$) is the set of all languages decided by a NDTM $\mathcal{M}$ in less than $g(n)$ steps (longer branch), with $g(n) \in \mathcal{O}(f(n))$

Proposition

- $\forall f : \mathbb{N} \mapsto \mathbb{N}$, then DTIME($f(n)$) $\subseteq$ NTIME($f(n)$)
- $\forall f(n) \geq n$, NTIME($f(n)$) is closed for finite union and finite intersection
  - if $\mathcal{L}_1, \ldots, \mathcal{L}_m \in$ NTIME($f(n)$), then $\mathcal{L}_1 \cup \cdots \cup \mathcal{L}_m \in$ NTIME($f(n)$)
  - if $\mathcal{L}_1, \ldots, \mathcal{L}_m \in$ NTIME($f(n)$), then $\mathcal{L}_1 \cap \cdots \cap \mathcal{L}_m \in$ NTIME($f(n)$)

Closeness under complement is an open question. The answer is mainly assumed to be « no »

Polynomial vs Exponential Time

Definition

- The complexity class NP is the set of languages decided in polynomial time by a NDTM, i.e

$$\mathrm{NP} = \bigcup_{k \in \mathbb{N}} \mathrm{NTIME}(n^k)$$

- The complexity class NEXP is the set of languages decided in exponential time by a NDTM, i.e

$$\mathrm{NEXP} = \bigcup_{k \in \mathbb{N}} \mathrm{NTIME}(2^{n^k})$$

Polynomial vs Exponential Time

Definition

- The complexity class NP is the set of languages decided in polynomial time by a NDTM, i.e

$$NP = \bigcup_{k \in \mathbb{N}} NTIME(n^k)$$

- The complexity class NEXP is the set of languages decided in exponential time by a NDTM, i.e

$$NEXP = \bigcup_{k \in \mathbb{N}} NTIME(2^{n^k})$$

Theorem

$$P \subseteq NP \subseteq EXP \subseteq NEXP$$

Moreover, $P \neq EXP$, $NP \neq NEXP$.
$P = NP$, $NP = EXP$ or $EXP = NEXP$ are *open questions*

Polynomial vs Exponential Time

Definition

- The complexity class NP is the set of languages decided in polynomial time by a NDTM, i.e

$$\mathrm{NP} = \bigcup_{k \in \mathbb{N}} \mathrm{NTIME}(n^k)$$

- The complexity class NEXP is the set of languages decided in exponential time by a NDTM, i.e

$$\mathrm{NEXP} = \bigcup_{k \in \mathbb{N}} \mathrm{NTIME}(2^{n^k})$$

Theorem

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{EXP} \subseteq \mathrm{NEXP}$$

Moreover, $\mathrm{P} \neq \mathrm{EXP}$, $\mathrm{NP} \neq \mathrm{NEXP}$.
$\mathrm{P} = \mathrm{NP}$, $\mathrm{NP} = \mathrm{EXP}$ or $\mathrm{EXP} = \mathrm{NEXP}$ are *open questions*: Millennium Prize Problems

Examples of Problems in NP

Clique

- Input: G a graph, $k \in \mathbb{N}$
- Problem: Does G contain a clique with size k?

Subset Sum

- Input: $\{a_1, \ldots, a_n\} \subset \mathbb{N}$, $k \in \mathbb{N}$
- Problem: Is there a subset $S \subseteq \{a_1, \ldots, a_n\}$ s.t. $\sum_{x \in S} x = k$?

Outline

Non-Deterministic Time Complexity Classes

Polynomial Hierarchy

Complexity of Well-Known Problems
- SAT and Related Problems
- Other Theoretical Problems
- Video Games

Determining the Complexity of a Problem

Complement of a Class

Definition

Given a complexity class C, its complement $\text{co}C$ is defined by

$$\text{co}C = \{\bar{\mathcal{P}} \mid \mathcal{P} \in C\}$$

For complexity classes C defined with DTM, $\text{co}C = C$

Complement of a Class

Definition

Given a complexity class C, its complement coC is defined by

$$\text{co}\mathsf{C} = \{\bar{\mathcal{P}} \mid \mathcal{P} \in \mathsf{C}\}$$

For complexity classes C defined with DTM, $\text{co}\mathsf{C} = \mathsf{C}$

Important Complement Class

coNP is the complement complexity class of NP

Examples of Problems in coNP

No Clique

- Input: G a graph, $k \in \mathbb{N}$
- Problem: Does G contain no clique with size k?

Why determining if a graph has a k-clique has not the same complexity than proving that it has no k-clique?

Examples of Problems in coNP

No Clique

- Input: G a graph, $k \in \mathbb{N}$
- Problem: Does G contain no clique with size k?

Why determining if a graph has a k-clique has not the same complexity than proving that it has no k-clique?

- To accept an instance of Clique: just exhibit one example of a k-clique to answer YES
- To accept an instance of No Clique: you have to check every k-subgraph G' and check if it's a clique

Relations between P, NP, $\textsc{co}$NP

Theorem

$$\mathrm{P} \subseteq \mathrm{NP} \text{ and } \mathrm{P} \subseteq \textsc{co}\mathrm{NP}$$

but $\mathrm{NP} = \textsc{co}\mathrm{NP}$ or $\mathrm{NP} \neq \textsc{co}\mathrm{NP}$ is still an open question

Relations between P, NP, coNP

Theorem

$$\mathrm{P} \subseteq \mathrm{NP} \text{ and } \mathrm{P} \subseteq \mathrm{coNP}$$

but NP $=$ coNP or NP $\neq$ coNP is still an open question

Idea of the polynomial hierarchy: define generalized complexity classes with similar inclusion pattern

Oracle Machines

Definition

Given C_1, C_2 two complexity classes, $C_1^{C_2}$ is the set of all problems which can be solved by a Turing machine from the class C_1 with an oracle from the class C_2

Oracle Machines

Definition

Given C_1, C_2 two complexity classes, $C_1^{C_2}$ is the set of all problems which can be solved by a Turing machine from the class C_1 with an oracle from the class C_2

Oracle of class C_2: abstract entity which can solve in one step a problem from C_2

Oracle Machines

Definition

Given C_1, C_2 two complexity classes, $C_1^{C_2}$ is the set of all problems which can be solved by a Turing machine from the class C_1 with an oracle from the class C_2

Oracle of class C_2: abstract entity which can solve in one step a problem from C_2

Example

A problem belongs to P^{NP} if it can be solved by a DTM with polynomially many calls to a NP oracle (i.e. a polynomial NDTM)

Polynomial Hierarchy
[Stockmeyer 1976]

Definition

The polynomial hierarchy is the set of complexity classes defined recursively by

- $\Delta_0^{\mathsf{P}} = \Sigma_0^{\mathsf{P}} = \Pi_0^{\mathsf{P}} = \mathsf{P}$
- $\Delta_{k+1}^{\mathsf{P}} = \mathsf{P}^{\Sigma_k^{\mathsf{P}}}$
- $\Sigma_{k+1}^{\mathsf{P}} = \mathsf{NP}^{\Sigma_k^{\mathsf{P}}}$
- $\Pi_{k+1}^{\mathsf{P}} = \mathsf{coNP}^{\Sigma_k^{\mathsf{P}}}$

Polynomial Hierarchy
[Stockmeyer 1976]

Definition

The polynomial hierarchy is the set of complexity classes defined recursively by

- $\Delta_0^P = \Sigma_0^P = \Pi_0^P = P$
- $\Delta_{k+1}^P = P^{\Sigma_k^P}$
- $\Sigma_{k+1}^P = NP^{\Sigma_k^P}$
- $\Pi_{k+1}^P = coNP^{\Sigma_k^P}$

P

Polynomial Hierarchy

[Stockmeyer 1976]

Definition

The polynomial hierarchy is the set of complexity classes defined recursively by

- $\Delta_0^P = \Sigma_0^P = \Pi_0^P = P$
- $\Delta_{k+1}^P = P^{\Sigma_k^P}$
- $\Sigma_{k+1}^P = NP^{\Sigma_k^P}$
- $\Pi_{k+1}^P = \text{co}NP^{\Sigma_k^P}$

$$\begin{tikzcd}
 & \text{co}NP \\
P \arrow[ur] \arrow[dr] & \\
 & NP
\end{tikzcd}$$

Polynomial Hierarchy

[Stockmeyer 1976]

Definition

The polynomial hierarchy is the set of complexity classes defined recursively by

- $\Delta_0^P = \Sigma_0^P = \Pi_0^P = P$
- $\Delta_{k+1}^P = P^{\Sigma_k^P}$
- $\Sigma_{k+1}^P = NP^{\Sigma_k^P}$
- $\Pi_{k+1}^P = \text{co}NP^{\Sigma_k^P}$

$$\begin{tikzcd}
 & \text{co}NP \arrow[dr] & \\
P \arrow[ur] \arrow[dr] & & \Delta_2^P \\
 & NP \arrow[ur] &
\end{tikzcd}$$

Polynomial Hierarchy
[Stockmeyer 1976]

Definition

The polynomial hierarchy is the set of complexity classes defined recursively by

- $\Delta_0^P = \Sigma_0^P = \Pi_0^P = P$
- $\Delta_{k+1}^P = P^{\Sigma_k^P}$
- $\Sigma_{k+1}^P = NP^{\Sigma_k^P}$
- $\Pi_{k+1}^P = \mathrm{coNP}^{\Sigma_k^P}$

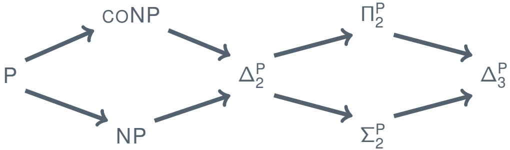
$$\begin{tikzcd}
& \mathrm{coNP} \arrow[dr] & & \Pi_2^P \\
P \arrow[ur] \arrow[dr] & & \Delta_2^P \arrow[ur] \arrow[dr] & \\
& NP \arrow[ur] & & \Sigma_2^P
\end{tikzcd}$$

Polynomial Hierarchy
[Stockmeyer 1976]

Definition

The polynomial hierarchy is the set of complexity classes defined recursively by

- $\Delta_0^P = \Sigma_0^P = \Pi_0^P = P$
- $\Delta_{k+1}^P = P^{\Sigma_k^P}$
- $\Sigma_{k+1}^P = NP^{\Sigma_k^P}$
- $\Pi_{k+1}^P = coNP^{\Sigma_k^P}$



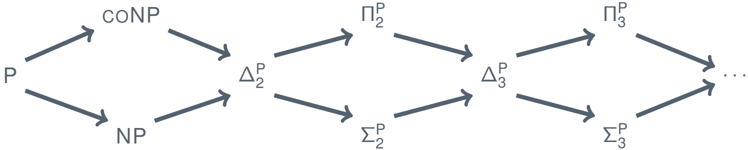
Polynomial Hierarchy

[Stockmeyer 1976]

Definition

The polynomial hierarchy is the set of complexity classes defined recursively by

- $\Delta_0^P = \Sigma_0^P = \Pi_0^P = P$
- $\Delta_{k+1}^P = P^{\Sigma_k^P}$
- $\Sigma_{k+1}^P = NP^{\Sigma_k^P}$
- $\Pi_{k+1}^P = coNP^{\Sigma_k^P}$



Definition

The polynomial hierarchy is the set of complexity classes defined recursively by

- $\Delta_0^P = \Sigma_0^P = \Pi_0^P = P$
- $\Delta_{k+1}^P = P^{\Sigma_k^P}$
- $\Sigma_{k+1}^P = NP^{\Sigma_k^P}$
- $\Pi_{k+1}^P = \mathrm{CO}NP^{\Sigma_k^P}$

$$
\begin{tikzcd}
 & \mathrm{CO}NP \arrow[dr] & & \Pi_2^P \arrow[dr] & & \Pi_3^P \arrow[dr] & \\
P \arrow[ur] \arrow[dr] & & \Delta_2^P \arrow[ur] \arrow[dr] & & \Delta_3^P \arrow[ur] \arrow[dr] & & \cdots \\
 & NP \arrow[ur] & & \Sigma_2^P \arrow[ur] & & \Sigma_3^P \arrow[ur] &
\end{tikzcd}
$$

- $C_1 \rightarrow C_2$ means that $C_1 \subseteq C_2$
- $PH = \bigcup_{i \in \mathbb{N}} \Sigma_i^P$

Relative Hardness of Problems

Polynomial-Time Functional Reduction

A polynomial-time functional reduction f is a total computable function from a problem $\mathcal{P}_1$ to a problem $\mathcal{P}_2$ such that, for any instance i of $\mathcal{P}_1$,

- $f(i)$ can be computed in polynomial-time in the size of i
- i is a positive instance of $\mathcal{P}_1$ iff $f(i)$ is a positive instance of $\mathcal{P}_2$

Notation:

$$\mathcal{P}_1 \leq_f^{\mathsf{P}} \mathcal{P}_2$$

Relative Hardness of Problems

Polynomial-Time Functional Reduction

A polynomial-time functional reduction f is a total computable function from a problem $\mathcal{P}_1$ to a problem $\mathcal{P}_2$ such that, for any instance i of $\mathcal{P}_1$,

- $f(i)$ can be computed in polynomial-time in the size of i
- i is a positive instance of $\mathcal{P}_1$ iff $f(i)$ is a positive instance of $\mathcal{P}_2$

Notation:

$$\mathcal{P}_1 \leq_f^{\mathsf{P}} \mathcal{P}_2$$

C-hardness

A problem $\mathcal{P}$ is C-hard iff for each $\mathcal{P}' \in \mathsf{C}$, $\mathcal{P}' \leq_f^{\mathsf{P}} \mathcal{P}$

Intuition: $\mathcal{P}$ is at least as hard to solve as every problem from C

Completeness

C-completeness

A problem $\mathcal{P}$ is C-complete iff it is C-hard and $\mathcal{P} \in$ C

Intuition: $\mathcal{P}$ is one of the hardest problems from C

Completeness

C-completeness

A problem $\mathcal{P}$ is C-complete iff it is C-hard and $\mathcal{P} \in$ C

Intuition: $\mathcal{P}$ is one of the hardest problems from C

A lot of interesting AI problems are complete for NP, coNP, Σ_2^P or Π_2^P

Outline

Non-Deterministic Time Complexity Classes

Polynomial Hierarchy

Complexity of Well-Known Problems
- SAT and Related Problems
- Other Theoretical Problems
- Video Games

Determining the Complexity of a Problem

- $V = \{x_1, \ldots, x_n\}$ a set of Boolean variables
- $C = \{\neg, \vee, \wedge\}$ a set of connectives
- A well formed formula (wff) ϕ is:
  - an atom: $\phi = x_i$
  - the negation of a wff: $\phi = \neg\psi$
  - the conjunction of two wffs: $\phi = \psi_1 \wedge \psi_2$
  - the disjunction of two wffs: $\phi = \psi_1 \vee \psi_2$
- Interpretation $\omega : V \mapsto \mathbb{B} = \{0, 1\}$
- Semantics of connectives:
  - $\omega(\neg\psi) = 1 - \omega(\psi)$
  - $\omega(\psi_1 \wedge \psi_2) = \min(\omega(\psi_1), \omega(\psi_2))$
  - $\omega(\psi_1 \vee \psi_2) = \max(\omega(\psi_1), \omega(\psi_2))$
- $\omega \models \phi$ iff $\omega(\phi) = 1$

Theorem [Cook 1971]

Given a propositional formula ϕ, the SAT problem consists in determining whether ϕ is consistent, i.e. whether ϕ has a model.

SAT is NP-complete.

General knowledge: SAT is the first problem which has been proved NP-complete.

Theorem [Cook 1971]

Given a propositional formula ϕ, the SAT problem consists in determining whether ϕ is consistent, i.e. whether ϕ has a model.

SAT is NP-complete.

General knowledge: SAT is the first problem which has been proved NP-complete.
The power of propositional logic to express a lot of « real » problems (solving games, planning,...) has led to the development of quite efficient methods to solve NP-complete problems. But even these methods do not allow to solve ALL instances of NP-complete problems.

Normal Forms

- A literal l is either a variable x or its negation $\neg x$
- A clause is a disjunction of literals $l_1 \vee \cdots \vee l_n$
- A cube is a conjunction of literals $l_1 \wedge \cdots \wedge l_n$

Conjunctive Normal Form

A formula is in CNF if it is a conjunction of clauses

Disjunctive Normal Form

A formula is in DNF if it is a disjunction of cubes

Complexity of SAT with Normal Forms

CNF-SAT

Any formula can be transformed in an equivalent CNF formula

- The transformation can be done in polynomial time

Complexity of SAT with Normal Forms

CNF-SAT

Any formula can be transformed in an equivalent CNF formula

- The transformation can be done in polynomial time
- Solving CNF-SAT is NP-complete

Complexity of SAT with Normal Forms

CNF-SAT

Any formula can be transformed in an equivalent CNF formula

- The transformation can be done in polynomial time
- Solving CNF-SAT is NP-complete

DNF-SAT

Any formula can be transformed in an equivalent DNF formula

- Solving DNF-SAT is polynomial

Complexity of SAT with Normal Forms

CNF-SAT

Any formula can be transformed in an equivalent CNF formula

- The transformation can be done in polynomial time
- Solving CNF-SAT is NP-complete

DNF-SAT

Any formula can be transformed in an equivalent DNF formula

- Solving DNF-SAT is polynomial
- The transformation cannot be done in polynomial time :(

- A binary clause is a clause with two literals: $l_1 \vee l_2$
- A 2CNF is a CNF formula with only binary clauses

Complexity of 2SAT

Determining if a 2CNF formula is satisfiable is in P

- A Horn clause is a clause with at most one positive literal: $x_1 \vee \neg x_2 \cdots \vee \neg x_n$
- A Horn formula (or Horn CNF) is a CNF formula with only Horn clauses

Complexity of Horn-SAT

Determining if a Horn formula is satisfiable is in P

Quantified Boolean Formula

- A canonical QBF is a formula $\mathcal{Q}_1\mathcal{X}_1, \mathcal{Q}_2\mathcal{X}_2, \dots \mathcal{Q}_n\mathcal{X}_n, \phi$ with
  - $\mathcal{Q}_i \in \{\forall, \exists\}$ and $\mathcal{Q}_i \neq \mathcal{Q}_{i+1}$
  - $\mathcal{X}_1, \dots \mathcal{X}_n$ form a partition of the Boolean variables in ϕ
  - ϕ is a propositional formula
- E.g. $\exists x_1, x_3, \forall x_2, (\neg x_1 \vee x_2) \wedge (\neg x_3 \vee x_2)$
- $\exists_n$QBF is the decision problem: is the QBF $\exists\mathcal{X}_1, \forall\mathcal{X}_2, \dots \mathcal{Q}_n\mathcal{X}_n, \phi$ true?
- $\forall_n$QBF is the decision problem: is the QBF $\forall\mathcal{X}_1, \exists\mathcal{X}_2, \dots \mathcal{Q}_n\mathcal{X}_n, \phi$ true?

Complexity of $\exists_n$QBF

$\exists_n$QBF is Σ_n^P-complete

Complexity of $\forall_n$QBF

$\forall_n$QBF is Π_n^P-complete

Set Packing

Definition

Given a universe $\mathcal{U}$ and $\mathcal{S} \subseteq 2^{\mathcal{U}}$, a set packing of $\mathcal{U}$ is a subset $\mathcal{C} \subseteq \mathcal{S}$ s.t. all elements in $\mathcal{C}$ are pairwise disjoints

Theorem [Karp 1972]

Given $\mathcal{U}, \mathcal{S}$ and $k \in \mathbb{N}$, determining whether there is a set packing $\mathcal{C}$ s.t. $|\mathcal{C}| = k$ is NP-complete

Knapsack Problem

Definition

Given a list of objects $x_1, \ldots, x_n$, each of them associated with a value $v_1, \ldots, v_n$ and a weight $w_1, \ldots, w_n$, a knapsack with a maximal weight W, and an integer V, is it possible to fill the bag with some of the objects, such that the sum of the weights is lesser than W and the sum of the values is greater than V?

Theorem

Solving the Knapsack Problem is NP-complete

Kernel of a Graph

Definition

A graph is a pair $G = \langle N, E \rangle$ where elements of N are called *nodes* and $E \subseteq N \times N$ is the set of *edges* between the nodes. A *kernel* of G is a subset $K \subseteq N$ s.t. $\forall n_i, n_j \in K, (n_i, n_j) \notin E$ and $\forall n_j \in N \setminus K$, $\exists n_i \in K$ s.t. $(n_i, n_j) \in E$

Theorem [Creignou 1995]

Given a graph G, determining whether G has a kernel is NP-complete.

Shortest Implicant

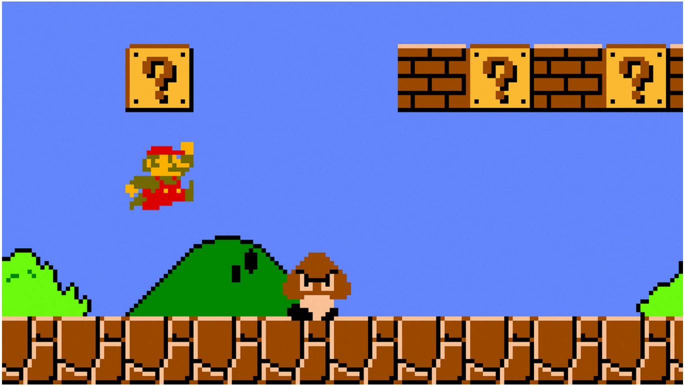
Definition

An implicant of a formula ϕ is a conjunction of literals $c = x_1 \wedge \cdots \wedge x_n$ s.t. $c \vdash \phi$.

Theorem [Umans 2001]

Given a formula ϕ and $k \in \mathbb{N}$, determining whether ϕ has an implicant c s.t. $|c| \leq k$ is Σ_2^P-complete

Super Mario Bros.



Theorem [Aloupis *et al.* 2015]

It is NP-hard to decide whether the goal is reachable from the start of a stage in generalized Super Mario Bros.

Donkey Kong Country

Theorem [Aloupis *et al.* 2015]

It is NP-hard to decide whether the goal is reachable from the start of a stage in generalized Donkey Kong Country.

The Legend of Zelda

Theorem [Aloupis *et al.* 2015]

It is NP-hard to decide whether a given target location is reachable from a given start location in generalized Legend of Zelda, LoZ II: The Adventure of Link and LoZ: A Link to the Past.

Metroid

Theorem [Aloupis *et al.* 2015]

It is NP-hard to decide whether a given target location is reachable from a given start location in generalized Metroid.

Pokémon

Theorem [Aloupis *et al.* 2015]

- It is NP-hard to decide whether a given target location is reachable from a given start location in generalized Pokémon.
- It is NP-complete to decide whether a given target location is reachable from a given start location in generalized Pokémon in which the only overworld game elements are enemy Trainers.

More Information on Complexity of Problems

- [Garey and Johnson 1979]: One of the most well-known book on the topic, a lot of classical results
- [Schaefer and Umans 2002a, Schaefer and Umans 2002b]: More recent collection of results

Outline

Non-Deterministic Time Complexity Classes

Polynomial Hierarchy

Complexity of Well-Known Problems
- SAT and Related Problems
- Other Theoretical Problems
- Video Games

Determining the Complexity of a Problem

Bounds of Complexity

In some cases, it's not easy to determine precisely the complexity of a problem, but we can give lower/upper bounds.

- Lower bound: C-hardness. E.g. if $\mathcal{P}$ is NP-hard, $\mathcal{P}$ is at least as hard as SAT
- Upper bound: C membership. E.g. if $\mathcal{P} \in \Sigma_2^{\mathsf{P}}$, $\mathcal{P}$ is at most as hard as Shortest Implicant problem.

Bounds of Complexity

In some cases, it's not easy to determine precisely the complexity of a problem, but we can give lower/upper bounds.

- Lower bound: C-hardness. E.g. if $\mathcal{P}$ is NP-hard, $\mathcal{P}$ is at least as hard as SAT
- Upper bound: C membership. E.g. if $\mathcal{P} \in \Sigma_2^{\mathsf{P}}$, $\mathcal{P}$ is at most as hard as Shortest Implicant problem.
- Exact complexity: C-completeness

Bounds of Complexity

In some cases, it's not easy to determine precisely the complexity of a problem, but we can give lower/upper bounds.

- Lower bound: C-hardness. E.g. if $\mathcal{P}$ is NP-hard, $\mathcal{P}$ is at least as hard as SAT
- Upper bound: C membership. E.g. if $\mathcal{P} \in \Sigma_2^{\mathsf{P}}$, $\mathcal{P}$ is at most as hard as Shortest Implicant problem.
- Exact complexity: C-completeness

Prove C-completeness: prove hardness (c.f. polynomial functional reductions) $+$ prove membership

Certificate: Intuition

- "NP algorithm" for SAT.

Algorithm 1 SAT

Input: ϕ

```
Let ω be some interpretation
for c a clause in ϕ do
    sat_clause = false
    for l a literal in c do
        if ω(l) = 1 then
            sat_clause = true
        end if
    end for
    if not sat_clause then
        return False
    end if
end for
return True
```

Certificate: Intuition

- “NP algorithm" for SAT.

Algorithm 2 SAT

```
Input: φ
  Let ω be some interpretation
  for c a clause in φ do
      sat_clause = false
      for l a literal in c do
          if ω(l) = 1 then
              sat_clause = true
          end if
      end for
      if not sat_clause then
          return False
      end if
  end for
  return True
```

- Non-deterministic guess

Certificate: Intuition

- "NP algorithm" for SAT.

Algorithm 3 SAT

```
Input: φ
  Let ω be some interpretation
  for c a clause in φ do
      sat_clause = false
      for l a literal in c do
          if ω(l) = 1 then
              sat_clause = true
          end if
      end for
      if not sat_clause then
          return False
      end if
  end for
  return True
```

- Non-deterministic guess
- Each execution of the algorithm tests a different value of ω

Certificate: Intuition

- "NP algorithm" for SAT.

Algorithm 4 SAT

Input: ϕ

```
Let ω be some interpretation
for c a clause in φ do
    sat_clause = false
    for l a literal in c do
        if ω(l) = 1 then
            sat_clause = true
        end if
    end for
    if not sat_clause then
        return False
    end if
end for
return True
```

- Non-deterministic guess
- Each execution of the algorithm tests a different value of ω
- If there is one execution that returns *True*, then ϕ is a positive instance

Certificate: Intuition

- "NP algorithm" for SAT.

Algorithm 5 SAT

Input: ϕ

```
Let ω be some interpretation
for c a clause in φ do
    sat_clause = false
    for l a literal in c do
        if ω(l) = 1 then
            sat_clause = true
        end if
    end for
    if not sat_clause then
        return False
    end if
end for
return True
```

- Non-deterministic guess
- Each execution of the algorithm tests a different value of ω
- If there is one execution that returns *True*, then ϕ is a positive instance
- In this case, ω is called a *certificate* for ϕ

Certificate

Definition

A certificate (also called a witness) is a word that certifies the answer to a computation, or certifies the membership of some word in a language.

Example

- $\mathcal{P} =$ "Given a polynomial P, has P at least one root?". The instance $P(x) = x^2$ can be verified with the certificate $x = 0$: $P(0) = 0$.
  $x = 0$ is a certificate that P is a positive instance of $\mathcal{P}$

Certificate

Definition

A certificate (also called a witness) is a word that certifies the answer to a computation, or certifies the membership of some word in a language.

Example

- $\mathcal{P}$ = "Given a polynomial P, has P at least one root?". The instance $P(x) = x^2$ can be verified with the certificate $x = 0$: $P(0) = 0$.
  $x = 0$ is a certificate that P is a positive instance of $\mathcal{P}$
- $\mathcal{P}$'= "Given a polynomial P, is $P(x)$ positive for all x?" The instance $P'(x) = x^2 - 2$ can be verified with the certificate $x = -1$: $P'(-1) = (-1)^2 - 2 = 1 - 2 = -1 < 0$.
  $x = -1$ is a certificate that P' is a negative instance of $\mathcal{P}$'

NP and coNP Membership

Proposition

Let $\mathcal{P}$ be a problem. Given an instance x of $\mathcal{P}$ and a certificate c, if the problem

$$\mathcal{P}': \text{« Is } c \text{ a proof that } x \text{ is a positive instance of } \mathcal{P}\text{? »}$$

is in P, then $\mathcal{P} \in$ NP

Proposition

Let $\mathcal{P}$ be a problem. Given an instance x of $\mathcal{P}$ and c a certificate, if the problem

$$\mathcal{P}': \text{« Is } c \text{ a proof that } x \text{ is a negative instance of } \mathcal{P}\text{? »}$$

is in P, then $\mathcal{P} \in$ coNP

NP and coNP Membership

Proposition

Let $\mathcal{P}$ be a problem. Given an instance *x* of $\mathcal{P}$ and a certificate *c*, if the problem

$$\mathcal{P}': \text{« Is } c \text{ a proof that } x \text{ is a positive instance of } \mathcal{P}\text{? »}$$

is in P, then $\mathcal{P} \in \text{NP}$

Proposition

Let $\mathcal{P}$ be a problem. Given an instance *x* of $\mathcal{P}$ and *c* a certificate, if the problem

$$\mathcal{P}': \text{« Is } c \text{ a proof that } x \text{ is a negative instance of } \mathcal{P}\text{? »}$$

is in P, then $\mathcal{P} \in \text{coNP}$

NP: Problems where checking a solution is easy
coNP: Problems where checking a counter-example is easy

Certificate Verification

- "NP algorithm" for SAT.

Algorithm 6 SAT

Input: ϕ

```
Let ω be some interpretation
for c a clause in φ do
    sat_clause = false
    for l a literal in c do
        if ω(l) = 1 then
            sat_clause = true
        end if
    end for
    if not sat_clause then
        return False
    end if
end for
return True
```

Certificate Verification

- "NP algorithm" for SAT.

Algorithm 8 SAT

Input: ϕ
  Let ω be some interpretation
  for c a clause in ϕ **do**
    $sat\_clause = false$
    for l a literal in c **do**
      if $\omega(l) = 1$ **then**
        $sat\_clause = true$
      end if
    end for
    if not sat_clause **then**
      return False
    end if
  end for
  return True

- "P algorithm" for verifying ω

Algorithm 9 Verify Interpretation

Input: ϕ, ω
  for c a clause in ϕ **do**
    $sat\_clause = false$
    for l a literal in c **do**
      if $\omega(l) = 1$ **then**
        $sat\_clause = true$
      end if
    end for
    if not sat_clause **then**
      return False
    end if
  end for
  return True

Certificate Verification

- "NP algorithm" for SAT.

Algorithm 10 SAT

Input: ϕ

```
  Let ω be some interpretation
  for c a clause in φ do
      sat_clause = false
      for l a literal in c do
          if ω(l) = 1 then
              sat_clause = true
          end if
      end for
      if not sat_clause then
          return False
      end if
  end for
  return True
```

- "P algorithm" for verifying ω

Algorithm 11 Verify Interpretation

Input: ϕ, ω

```
  for c a clause in φ do
      sat_clause = false
      for l a literal in c do
          if ω(l) = 1 then
              sat_clause = true
          end if
      end for
      if not sat_clause then
          return False
      end if
  end for
  return True
```

General C-Membership with a Certificate

Proposition

Let $\mathcal{P}$ be a problem. Given an instance x of $\mathcal{P}$ and a certificate c, if the problem

$\mathcal{P}'$: « Is c a proof that x is a positive instance of $\mathcal{P}$? »

is in Π^{P}_{i-1}, then $\mathcal{P} \in \Sigma^{\mathsf{P}}_{i}$

General C-Membership with a Certificate

Proposition

Let $\mathcal{P}$ be a problem. Given an instance x of $\mathcal{P}$ and a certificate c, if the problem

$$\mathcal{P}': \text{« Is } c \text{ a proof that } x \text{ is a positive instance of } \mathcal{P}\text{? »}$$

is in Π^{P}_{i-1}, then $\mathcal{P} \in \Sigma^{\mathsf{P}}_{i}$

Proposition

Let $\mathcal{P}$ be a problem. Given an instance x of $\mathcal{P}$ and a certificate c, if the problem

$$\mathcal{P}': \text{« Is } c \text{ a proof that } x \text{ is a negative instance of } \mathcal{P}\text{? »}$$

is in $\Sigma^{\mathsf{P}}_{i-1}$, then $\mathcal{P} \in \Pi^{\mathsf{P}}_{i}$

References

[Turing 1936] A. M. Turing, *On computable numbers, with an application to the Entscheidungsproblem*. Proceedings of the London Mathematical Society, 1936.

[Cook 1972] S. A. Cook, *A Hierarchy for Nondeterministic Time Complexity*. STOC, 187–192, 1972.

[Seiferas et al 1978] J. I. Seiferas, M. J. Fischer and A. R. Meyer, *Separating Non-deterministic Time Complexity Classes*. J. ACM, 25.1, 146–167, 1978.

[Stockmeyer 1976] L. J. Stockmeyer, *The polynomial-time hierarchy*. Theoretical Computer Science, 3, 1–22, 1976.

References

[Cook 1971] S. A. Cook, *The Complexity of Theorem-Proving Procedures*. Proc. of the Third Annual Symposium on Theory of Computing, 151–158, 1971.

[Karp 1972] R. M. Karp, *Reducibility Among Combinatorial Problems*. Proc. of Symposium on the Complexity of Computer Computations, 85–103, 1972.

[Creignou 1995] N. Creignou, *The Class of Problems That are Linearly Equivalent to Satisfiability or a Uniform Method for Proving NP-Completeness*. Theoretical Computer Science, 145, 111–145, 1995.

[Umans 2001] C. Umans, *The Minimum Equivalent DNF Problem and Shortest Implicants*. J. Comput. Syst. Sci., 63, 597–611, 2001.

References

[Aloupis *et al.* 2015] G. Aloupis, E. D. Demaine, A. Guo and G. Viglietta, *Classic Nintendo games are (computationally) hard*. Theor. Comput. Sci. 586: 135-160, 2015.

[Garey and Johnson 1979] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[Schaefer and Umans 2002a] M. Schaefer and C. Umans, *Completeness in the Polynomial-Time Hierarchy: A Compendium*. Sigact News September, 2002.

[Schaefer and Umans 2002b] M. Schaefer and C. Umans, *Completeness in the Polynomial-Time Hierarchy: Part II*. Sigact News December, 2002.