Algorithms for Computational Logic

Introduction

Emmanuel Hebrard (adapted from **João Marques Silva**, Inês Lynce and Vasco Manquinho)

Outline

1. **The Complexity of SAT**
2. **The Tractability of SAT Fragments**

Outline

1. **The Complexity of SAT**
   - **P** and **NP**
   - Cook-Levin Theorem
2. **The Tractability of SAT Fragments**
   - Tractable Fragments

P vs. NP

Cook-Levin Theorem

SAT is **NP**-complete

- SAT is "*at least as hard*" as **any** problem in **NP**
  - If there exists a polynomial algorithm for SAT then there exists one for every problem in **NP**
  - If $SAT \in \mathbf{P}$ then $\mathbf{NP} = \mathbf{P}$

- Recall:

P

Set of problems that are solved by a *polynomial Turing Machine* (running in $\mathcal{O}(n^c)$ time for a constant c)

NP

Set of problems that are solved by a polynomial *Non-determinist* Turing Machine (running in $\mathcal{O}(n^c)$ time for a constant c)

NP-hard problem

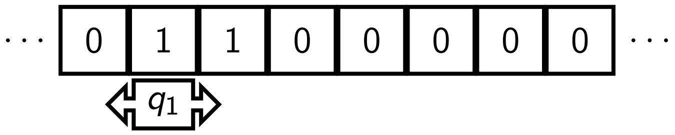
A problem Q is $\mathbf{NP}$-hard if it is "at least as hard as the hardest problem in $\mathbf{NP}$": if Q can be solved in $\mathcal{O}(T)$ time then any problem in $\mathbf{NP}$ can be solved in $\mathcal{O}(Tn^c)$ time for some constant c.

- If an $\mathbf{NP}$-hard problem can be solved in polynomial time, then $\mathbf{P} = \mathbf{NP}$

NP-complete problem

A problem Q is $\mathbf{NP}$-complete if it is $\mathbf{NP}$-hard and is in $\mathbf{NP}$

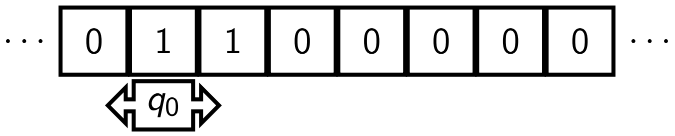
Turing Machines



Infinite tape

- An infinite **tape**, where we can read/write the symbols **0** and **1** and a **head**
- A "program"
  - A finite set of **states** with an initial state q_0 and a final state q_f.
  - A **transition table** associating a triplet $\langle$ state, symbol, $\{\leftarrow, \rightarrow\}$ $\rangle$ to every pair $\langle$ state, symbol $\rangle$
- Meaning: "if reading symbol x in state q then write x', change to state q' and move right/left"

Turing Machines, example



Infini tape

état	symbol 0	symbol 1
q_0	$q_f, 0, *$	$q_1, 0, \rightarrow$
q_1	$q_2, 0, \rightarrow$	$q_1, 1, \rightarrow$
q_2	$q_3, 1, \leftarrow$	$q_2, 1, \rightarrow$
q_3	$q_4, 0, \leftarrow$	$q_3, 1, \leftarrow$
q_4	$q_0, 1, \rightarrow$	$q_4, 1, \leftarrow$

Non-determinist Turing Machines

- A non-determinist Turing Machine can have several transitions in the same configuration
- We assume that it makes the right choice (or explore all possible choices in parallel)
- It is sufficient to have up two transitions for any one configuration

état	symbol 0	symbol 1
q_0	$q_f, 0, *$	$q_1, 0, \rightarrow$
q_1	$q_2, 0, \rightarrow$ ou $q_4, 1, \leftarrow$	$q_1, 1, \rightarrow$
q_2	$q_3, 1, \leftarrow$	$q_2, 1, \rightarrow$
q_3	$q_4, 0, \leftarrow$	$q_3, 1, \leftarrow$
q_4	$q_0, 1, \rightarrow$	$q_4, 1, \leftarrow$

Proof of the Cook-Levin theorem (1)

- Consider a problem Q and a Turing machine that solves it in polynomial time: $\mathcal{O}(n^c)$ pour une donnée de taille n
- This machine executes $\mathcal{O}(n^c)$ instructions and therefore requires a tape of length $\mathcal{O}(n^c)$
- We build the propositional logic formula with the following variables:
  - A variable $R_{i,t}$ for every cell i of the tape, every symbol k and every time step t: **true iff the symbol v written on cell** i **at time** t **is** k ($\mathcal{O}(1)$ symbols, hence $\mathcal{O}(n^{2c})$ variables)
  - A variable $L_{i,t}$ for every cell i of the tape and every time step t: **true iff the head is at position** i **at time** t ($\mathcal{O}(n^{2c})$ variables)
  - A variable $Q_{j,t}$ for every state q_j of the program and every time step t: **true iff the machine is in state** q_j **at time** t ($\mathcal{O}(1)$ states, hence $\mathcal{O}(n^c)$ variables)

état	symbol 00	symbol 1
q_0	$q_f, 0, *$	$q_1, 0, \rightarrow$
q_1	$q_2, 0, \rightarrow$	$q_1, 1, \rightarrow$
$q_2 q_2$	$q_3, 1, \leftarrow q_3, 1, \leftarrow$	$q_2, 1, \rightarrow$
q_3	$q_4, 0, \leftarrow$	$q_3, 1, \leftarrow$
q_4	$q_0, 1, \rightarrow$	$q_4, 1, \leftarrow$

- For a transition $(q_2, 0) \implies (q_3, 1, \leftarrow)$, we add the following clauses, for all i and all t:
  - $Q_{2,t} \wedge L_{i,t} \wedge R_{i,0,t} \Rightarrow Q_{3,t+1}$
  - $Q_{2,t} \wedge L_{i,t} \wedge R_{i,0,t} \Rightarrow L_{i-1,t+1}$
  - $Q_{2,t} \wedge L_{i,t} \wedge R_{i,0,t} \Rightarrow R_{i,1,t+1}$
- $\Theta(n^c)$ other clauses

Proof of the Cook-Levin theorem (2)

- Consider a problem $Q \in \mathbf{P}$
- Q admits a Turing machine that runs in $\mathcal{O}(|x|^{c_1})$ time
- For any input x, there exists a Horm Forumla $\phi(Q, x)$ such that:
  - $\phi(Q, x)$ is satisfiable if and only if $Q(x) = \mathbf{true}$
  - $|\phi(Q, x)| \in \mathcal{O}(|x|^{c_2})$
- An algorithm for *Horn*-SAT can solve any problem in $\mathbf{P}$ in polynomial time
  - Not so useful in itself (though *Horn*-SAT is $\mathbf{P}$-complete for log space reductions)

Proof of the Cook-Levin theorem (3)

- Can we come up with a similar encoding for *non-deterministic* machines ?

état	symbol 0	symbol 1
q_0	$q_f, 0, *$	$q_1, 0, \rightarrow$
q_1	$q_2, 0, \rightarrow$	$q_1, 1, \rightarrow$
q_2	$q_3, 1, \leftarrow$ $q_4, 0, \rightarrow$	$q_2, 1, \rightarrow$
q_3	$q_4, 0, \leftarrow$	$q_3, 1, \leftarrow$
q_4	$q_0, 1, \rightarrow$	$q_4, 1, \leftarrow$

- There are $\mathcal{O}(1)$ non-deterministic transitions (in the program)
- We add a variable $X_{l,t}$ for every non-deterministic transition l and for every time t
- The transition clauses become:
  - $X_{l,t} \wedge Q_{2,t} \wedge L_{i,t} \wedge R_{i,0,t} \Rightarrow Q_{3,t+1}$
  - $X_{l,t} \wedge Q_{2,t} \wedge L_{i,t} \wedge R_{i,0,t} \Rightarrow L_{i-1,t+1}$
  - $X_{l,t} \wedge Q_{2,t} \wedge L_{i,t} \wedge R_{i,0,t} \Rightarrow R_{i,1,t+1}$
  - $\neg X_{l,t} \wedge Q_{2,t} \wedge L_{i,t} \wedge R_{i,0,t} \Rightarrow Q_{4,t+1}$
  - $\neg X_{l,t} \wedge Q_{2,t} \wedge L_{i,t} \wedge R_{i,0,t} \Rightarrow L_{i+1,t+1}$
  - $\neg X_{l,t} \wedge Q_{2,t} \wedge L_{i,t} \wedge R_{i,0,t} \Rightarrow R_{i,0,t+1}$
- They are not Horn anymore
  Otherwise we would have shown $\mathbf{P} = \mathbf{NP}$!

Proof of the Cook-Levin theorem (conclusion)

Preuve

- Consider a problem $Q \in \mathbf{P}$
- Q admits a non-determinist Turing machine that runs in $\mathcal{O}(|x|^{c_1})$ time
- For any input x there exists a Boolean formula $\phi(Q, x)$ such that:
  - $\phi(Q, x)$ is satisfiable if and only if $x \in \mathbf{true}(Q)$ et $|\phi(Q, x)| \in \mathcal{O}(|x|^{c_2})$
- All problems in $\mathbf{NP}$ reduce to SAT
  - If SAT is in $\mathbf{P}$, then all problems in $\mathbf{NP}$ can be solved in polynomial time and therefore $\mathbf{P} = \mathbf{NP}$
  - If SAT is not in $\mathbf{P}$, then $\mathbf{P} \neq \mathbf{NP}$

- Si $SAT \in \mathbf{P}$ alors on peut trouver une interprétation de $\phi(Q, x)$ en temps polynomial, et donc résoudre Q en temps polynomial, quel que soit $Q \in \mathbf{NP}$
- Donc $SAT \in \mathbf{P}$ implique $\mathbf{P} = \mathbf{NP}$!

Outline

1 The Complexity of SAT
- $\mathbf{P}$ and $\mathbf{NP}$
- Cook-Levin Theorem

2 The Tractability of SAT Fragments
- Tractable Fragments

k-SAT

- SAT is **NP**-complete (Cook's theorem)
- 3-SAT is hard: **Exercise**
  - Encoding:

$$(p_1 \vee p_2 \vee x) \wedge (\neg x \vee p_3 \vee \ldots \vee p_k) \iff (p_1 \vee p_2 \vee \ldots \vee p_k)$$

- 2-SAT is easy (Resolution)
- *Horn*-SAT is easy (Unit propagation)

Intermediate Problems

Ladner's Theorem

If $\mathbf{P} = \mathbf{NP}$, then there are problems in $\mathbf{NP}$ that are neither in $\mathbf{P}$ nor $\mathbf{NP}$-complete.

- For instance GRAPHISOMORPHISM may be such problem; or FACTORISATION
- What about fragments of SAT?
  - We know some are easy (2-SAT, *Horn*-SAT), are there others?
  - How do we know which ones are hard and which ones are easy?
  - Are there some in the intermediate class?

Constraint Satisfaction Problems (reminder)

Constraint Satisfaction Problem (CSP)

Data: a triplet $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ where:

- $\mathcal{X}$ is a ordered set of *variables*
- $\mathcal{D}$ is a *domain*
- $\mathcal{C}$ is a set of *constraints*, where for $c \in \mathcal{C}$:
  - its *scope* $S(c)$ is a list of variables
  - its *relation* $R(c)$ is a subset of $\mathcal{D}^{|S(c)|}$

Question: does there exist a solution $\sigma \in \mathcal{D}^{|\mathcal{X}|}$ such that for every $c \in \mathcal{C}$, $\sigma(S(c)) \in R(c)$?

Projection

The projection $\sigma(X)$ of a tuple σ on a set of variables $X = (x_{i_1}, \ldots, x_{i_k}) \subseteq \mathcal{X}$ as the tuple $(\sigma(x_{i_1}), \ldots, \sigma(x_{i_k}))$

- Example: the constraint $x + y = z$ (on the Boolean ring)

x	y	z	$S(x+y=z)$
0	0	0	$R(x+y=z)$
0	1	1	
1	0	1	
1	1	0	

CNF and Generalized Relations

- A relation $R(c)$ over some variables can easily be expressed in clausal form
- Each clause excludes exactly one tuple, example: $x + y + z \neq 2$

$x+y+z \neq 2$			$x+y+z=2$			$\iff$	CNF
x	y	z	x	y	z		
0	0	0	0	1	1	$(\bar{x} \wedge y \wedge z) \vee$	$(x \vee \bar{y} \vee \bar{z}) \wedge$
0	0	1	1	0	1	$(x \wedge \bar{y} \wedge z) \vee$	$(\bar{x} \vee y \vee \bar{z}) \wedge$
0	1	0	1	1	0	$(x \wedge y \wedge \bar{z}) \vee$	$(\bar{x} \vee \bar{y} \vee z) \wedge$
1	0	0					
1	1	1					

- A clause is a particular case of relation on the Boolean domain

CSP Fragments

- We can define fragments of CSP via restrictions on the *domain*, the *structure* or on the *language*
  - **Domain**: Boolean CSPs: $\mathcal{D} = \{0, 1\}$, Three-valued CSPs, CSP on $\mathbb{Z}$, etc.
  - **Structure**: e.g., the incidence graph (bipartite graph variables / constraints) is a *tree* or has a bounded *treewidth*
  - **Language**: the library of relations is restricted to a given set Γ

Language fragment

CSP(Γ) is the problem of deciding the satisfiability of a CSP whose constraints all have relations in Γ.

- For instance Three-valued CSP($\{\neq\}$) is $\mathbf{NP}$-hard since 3-COLORATION is $\mathbf{NP}$-hard

Definability

pp-definability

A relation R over $x_1, \ldots, x_k$ on domain $\mathcal{D}$ is *(pp-)definable* from a set of relation Γ if and only if there exists a CSP $\mathcal{N} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ such that:

- $\{x_1, \ldots, x_k\} \subseteq \mathcal{X}$
- $c \in \mathcal{C} \implies R(c) \in \Gamma \cup \{=\}$
- $R(x_1, \ldots, x_k) \iff (x_1, \ldots, x_k)$ can be extended to a solution of $\mathcal{N}$

- i.e., the relation R can be encoded using relations in Γ
  - $<$ is definable from $\{\leq, \neq\}$
  - A k-clause $(p_1 \vee \ldots \vee p_k)$ is definable from 3-clauses
  - All k-ary relations are definable from k-clauses

Closure

$\ll \Gamma \gg$ is the set of relations that are definable from Γ

- CSP(Γ) and CSP($\ll \Gamma \gg$) have the same complexity
- Boolean CSPs whose incidence graph is such that constraints vertices have degree 2 (constraints are on at most 2 variables) is in $\mathbf{P}$
  - Any binary relation is definable by binary clauses
  - If Γ is the languages composed of 2-clauses, $\{(x \vee y), (\bar{x} \vee y), (\bar{x} \vee \bar{y})\}$, then:
    - ★ CSP(Γ) is $2-SAT$
    - ★ CSP($\ll \Gamma \gg$) is "Boolean binary CSP"

Schaefer's Theorem

Boolean CSP($\ll \Gamma \gg$) is in $\mathbf{P}$ if:

- Γ are 2-clauses
- Γ are Horn-clauses
- Γ are dual Horn-clauses
- $\Gamma = \{\oplus\}$ (i.e., XOR. Also known as "AFFINE-SAT")
- Every relation in Γ accepts the tuple with only 0
- Every relation in Γ accepts the tuple with only 1

and is $\mathbf{NP}$-hard otherwise

- *Dichotomy:* we know the complexity of *all* the language-based fragments of SAT, and none of them is an intermediate problem