Technische Universität Dresden

International Center for Computational Logic

COMPLEXITY THEORY

Lecture 7: NP-Completeness

Sergei Obiedkov

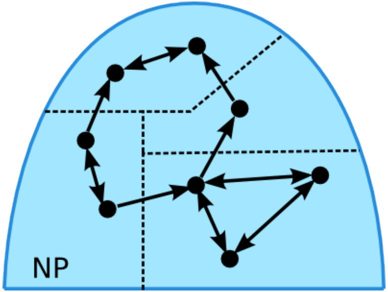
Knowledge-Based Systems

TU Dresden, 3 Nov 2024

Are NP Problems Hard?

The Structure of NP

Idea: polynomial many-one reductions define an order on problems



NP-Hardness and NP-Completeness

Definition 7.1:

(1) A language **H** is NP-hard, if **L** $\leq_p$ **H** for every language **L** $\in$ NP.

(2) A language **C** is NP-complete, if **C** is NP-hard and **C** $\in$ NP.

NP-Completeness

- NP-complete problems are the hardest problems in NP.
- They constitute the maximal class (wrt. $\leq_p$) of problems within NP.
- They are all equally difficult: an efficient solution to one would solve them all.

Theorem 7.2: If **L** is NP-hard and **L** $\leq_p$ **L**$'$, then **L**$'$ is NP-hard as well.

Proving NP-Completeness

How to show NP-completeness

To show that **L** $\in$ NP is NP-complete, we must show that every language in NP can be reduced to **L** in polynomial time.

Alternative approach

Given an NP-complete language **C**, we can show that another language **L** is NP-complete just by showing that

- **C** $\leq_p$ **L**
- **L** $\in$ NP

However: Is there any NP-complete problem at all?

Yes, thousands of them!

The Cook–Levin Theorem

Theorem 7.3 (Cook 1970, Levin 1973): SAT is NP-complete.

Proof:

(1) **SAT** $\in$ NP

Take satisfying assignments as polynomial certificates for the satisfiability of a formula.

(2) **SAT** is hard for NP

Proof by reduction from any word problem of some polynomially time-bounded NTM.

– For this proof, we assume that **SAT** is the satisfiability of an arbitrary propositional-logic formula rather than that of a CNF.

□

Proving the Cook–Levin Theorem: Main Objective

Given:

- a polynomial p
- a p-time bounded 1-tape NTM $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}})$
- a word w

Intended reduction: Define a propositional-logic formula $\varphi_{p,\mathcal{M},w}$ such that

(1) $\varphi_{p,\mathcal{M},w}$ is satisfiable if and only if $\mathcal{M}$ accepts w in time $p(|w|)$

(2) $\varphi_{p,\mathcal{M},w}$ is polynomial with respect to $|w|$

Proving the Cook–Levin Theorem: Rationale

Given: polynomial p, NTM $\mathcal{M}$, word w

Intended reduction: Define a propositional logic formula $\varphi_{p,\mathcal{M},w}$ such that

(1) $\varphi_{p,\mathcal{M},w}$ is satisfiable if and only if $\mathcal{M}$ accepts w in time $p(|w|)$

(2) $\varphi_{p,\mathcal{M},w}$ is polynomial with respect to $|w|$

Why does this prove NP-hardness of **Sat**?

It leads to a reduction $\mathbf{L} \leq_p$ **Sat** for every language $\mathbf{L} \in$ NP:

- If $\mathbf{L} \in$ NP, then there is an NTM $\mathcal{M}$ that is time-bounded by some polynomial p, such that $\mathbf{L}(\mathcal{M}) = \mathbf{L}$.
- The function $f_{\mathcal{M},p} : w \mapsto \varphi_{p,\mathcal{M},w}$ shows $\mathbf{L} \leq_p$ **Sat**:
  - f is a many-one reduction due to item (1) above
  - f is polynomial due to item (2) above

Note: We do not claim the transformation $\langle p, \mathcal{M}, w\rangle \mapsto \varphi_{p,\mathcal{M},w}$ to be polynomial in the size of p, $\mathcal{M}$, and w. Indeed, this would not hold true under reasonable encodings of p. However, being (multi-)exponential in p is not a concern, since the many-one reductions $f_{\mathcal{M},p}$ each use a fixed p and only care about the asymptotic complexity as w grows.

Proving Cook–Levin: Encoding Configurations

Idea: Use logic to describe a run of $\mathcal{M}$ on input w by a formula.

Note: On input w of length $n := |w|$, every computation path of $\mathcal{M}$ is of length $\leq p(n)$ and uses $\leq p(n)$ tape cells.

Use propositional variables for describing configurations:

- Q_s for each $s \in Q$ means "$\mathcal{M}$ is in state $s \in Q$"
- P_i for each $0 \leq i \leq p(n)$ means "the head is at Position i"
- $S_{a,i}$ for each $a \in \Gamma$ and $0 \leq i \leq p(n)$ means "tape cell i contains Symbol a"

Represent configuration $(q, hp, a_0 \dots a_{p(n)})$ by truth assignments to variables from the set

$$\overline{C} := \{Q_s, P_i, S_{a,i} \mid s \in Q, \quad a \in \Gamma, \quad 0 \leq i \leq p(n)\}$$

using the truth assignment β defined as

$$\beta(Q_s) := \begin{cases} 1 & s = q \\ 0 & s \neq q \end{cases} \qquad \beta(P_i) := \begin{cases} 1 & i = hp \\ 0 & i \neq hp \end{cases} \qquad \beta(S_{a,i}) := \begin{cases} 1 & a = a_i \\ 0 & a \neq a_i \end{cases}$$

Proving Cook–Levin: Validating Configurations

We define a formula Conf($\overline{C}$) for a set of configuration variables

$$\overline{C} = \{Q_s, P_i, S_{a,i} \mid s \in Q, \quad a \in \Gamma, \quad 0 \leq i \leq p(n)\}$$

as follows:

Conf($\overline{C}$) :=	"the assignment is a valid configuration":
$\bigvee_{q \in Q} \Big(Q_q \wedge \bigwedge_{s \in Q \setminus \{q\}} \neg Q_s \Big)$	"TM in exactly one state $q \in Q$"
$\wedge \bigvee_{0 \leq hp \leq p(n)} \Big(P_{hp} \wedge \bigwedge_{\substack{0 \leq i \leq p(n) \\ i \neq hp}} \neg P_i \Big)$	"head in exactly one position $0 \leq hp \leq p(n)$"
$\wedge \bigwedge_{0 \leq i \leq p(n)} \bigvee_{a \in \Gamma} \Big(S_{a,i} \wedge \bigwedge_{a \neq b \in \Gamma} \neg S_{b,i} \Big)$	"exactly one $a \in \Gamma$ in each cell"

Proving Cook–Levin: Validating Configurations

For an assignment β defined on variables in $\overline{C}$ define

$$\mathrm{conf}(\overline{C},\beta) := \left\{ (q, hp, w_0 \dots w_{p(n)}) \;\middle|\; \begin{array}{l} \beta(Q_q) = 1, \\ \beta(P_{hp}) = 1, \\ \beta(S_{w_i,i}) = 1 \text{ for all } 0 \leq i \leq p(n) \end{array} \right\}$$

Note: β may be defined on other variables besides those in $\overline{C}$.

> **Lemma 7.4:** If β satisfies $\mathrm{Conf}(\overline{C})$ then $|\mathrm{conf}(\overline{C},\beta)| = 1$.
> We can therefore write $\mathrm{conf}(\overline{C},\beta) = (q, hp, w)$ to simplify notation.

Observations:

- $\mathrm{conf}(\overline{C},\beta)$ is a potential configuration of $\mathcal{M}$, but it may not be reachable from the start configuration of $\mathcal{M}$ on input w.
- Conversely, every configuration $(q, hp, w_1 \dots w_{p(n)})$ induces a satisfying assignment β or which $\mathrm{conf}(\overline{C},\beta) = (q, hp, w_1 \dots w_{p(n)})$.

Proving Cook–Levin: Transitions Between Configurations

Consider the following formula $\mathsf{Next}(\overline{C}, \overline{C}')$ defined as

$$\mathsf{Conf}(\overline{C}) \wedge \mathsf{Conf}(\overline{C}') \wedge \mathsf{NoChange}(\overline{C}, \overline{C}') \wedge \mathsf{Change}(\overline{C}, \overline{C}').$$

$$\mathsf{NoChange} := \bigvee_{0 \leq hp < p(n)} \Big(P_{hp} \wedge \bigwedge_{\substack{i \neq hp \\ a \in \Gamma}} (S_{a,i} \to S'_{a,i}) \Big)$$

$$\mathsf{Change} := \bigvee_{0 \leq hp < p(n)} \Big(P_{hp} \wedge \bigvee_{\substack{q \in Q \\ a \in \Gamma}} \big(Q_q \wedge S_{a,hp} \wedge \bigvee_{(q',b,D) \in \delta(q,a)} (Q'_{q'} \wedge S'_{b,hp} \wedge P'_{D(hp)}) \big) \Big)$$

where $D(hp)$ is the position reached by moving in direction D from hp.

Lemma 7.5: For any assignment β defined on $\overline{C} \cup \overline{C}'$:

β satisfies $\mathsf{Next}(\overline{C}, \overline{C}')$ if and only if $\mathsf{conf}(\overline{C}, \beta) \vdash_{\mathcal{M}} \mathsf{conf}(\overline{C}', \beta)$

Proving Cook–Levin: Start and End

Defined so far:

- Conf($\overline{C}$): $\overline{C}$ describes a potential configuration
- Next($\overline{C}, \overline{C}'$): conf($\overline{C}, \beta$) $\vdash_{\mathcal{M}}$ conf($\overline{C}', \beta$)

Start configuration: For an input word $w = w_0 \cdots w_{n-1} \in \Sigma^*$, we define:

$$\text{Start}_{\mathcal{M},w}(\overline{C}) := \text{Conf}(\overline{C}) \wedge Q_{q_0} \wedge P_0 \wedge \bigwedge_{i=0}^{n-1} S_{w_i,i} \wedge \bigwedge_{i=n}^{p(n)} S_{\llcorner\!\lrcorner,i}$$

Then an assignment β satisfies Start$_{\mathcal{M},w}(\overline{C})$ if and only if $\overline{C}$ represents the start configuration of $\mathcal{M}$ on input w.

Accepting stop configuration:

$$\text{Acc-Conf}(\overline{C}) := \text{Conf}(\overline{C}) \wedge Q_{q_\text{accept}}$$

Then an assignment β satisfies Acc-Conf($\overline{C}$) if and only if $\overline{C}$ represents an accepting configuration of $\mathcal{M}$.

Proving Cook–Levin: Adding Time

Since $\mathcal{M}$ is p-time bounded, each run may contain up to $p(n)$ steps
$\leadsto$ we need one set of configuration variables for each step

Propositional variables:

$Q_{q,t}$ for all $q \in Q$, $0 \leq t \leq p(n)$ means "at time t, $\mathcal{M}$ is in state $q \in Q$"

$P_{i,t}$ for all $0 \leq i, t \leq p(n)$ means "at time t, the head is at position i"

$S_{a,i,t}$ for all $a \in \Gamma$ and $0 \leq i, t \leq p(n)$ means "at time t, tape cell i contains symbol a"

Notation:

$$\overline{C}_t := \{Q_{q,t},\ P_{i,t},\ S_{a,i,t} \quad | \quad q \in Q, \quad 0 \leq i \leq p(n), \quad a \in \Gamma\}$$

Proving Cook–Levin: The Formula

Given:

- a polynomial p
- a p-time bounded 1-tape NTM $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}})$
- a word w

We define the formula $\varphi_{p,\mathcal{M},w}$ as follows:

$$\varphi_{p,\mathcal{M},w} := \text{Start}_{\mathcal{M},w}(\overline{C}_0) \wedge \bigvee_{0 \leq t \leq p(n)} \left(\text{Acc-Conf}(\overline{C}_t) \wedge \bigwedge_{0 \leq i < t} \text{Next}(\overline{C}_i, \overline{C}_{i+1}) \right)$$

"C_0 encodes the start configuration", and, for some polynomial time t:

"$\mathcal{M}$ accepts after t steps" and "$\overline{C}_0, \ldots, \overline{C}_t$ encode a computation path"

Lemma 7.6: $\varphi_{p,\mathcal{M},w}$ is satisfiable if and only if $\mathcal{M}$ accepts w in time $p(|w|)$.

Note that an accepting or rejecting stop configuration has no successor.

Lemma 7.7: The size of $\varphi_{p,\mathcal{M},w}$ is polynomial in $|w|$.

The Cook–Levin Theorem

Theorem 7.3 (Cook 1970, Levin 1973): Sat is NP-complete.

Proof:

(1) Sat $\in$ NP

Take satisfying assignments as polynomial certificates for the satisfiability of a formula.

(2) Sat is hard for NP

Proof by reduction from any word problem of some polynomially time-bounded NTM.

– For this proof, we assume that **Sat** is the satisfiability of an arbitrary propositional-logic formula rather than that of a CNF.

□

SAT as a Basic NP-complete Problem

Sat-Solvers

- Any problem in NP can be solved by reducing it to **Sat** (in polynomial time) and then solving the resulting **Sat** problem.
- There are algorithms that can process real-life instances of **Sat** with hundreds and even thousands of variables.
- There are annual competitions where the best algorithms are announced.

- PySAT: a Python toolkit providing an interface to several **Sat** solvers `https://pysathq.github.io`

- Get yourself familiar with PySAT: `https://tinyurl.com/7kba64n9`

An NP Problem: CLUSTERING

- Partition items (e.g., text documents) so that similar ones (e.g., in topics covered) are grouped together and dissimilar ones are separated.
- One possible formalization

Input:
I is the set of items to be clustered
k is the desired number of clusters
D is the (symmetric) matrix of distances between items in I
$min\_spacing$ is the smallest allowed distance between items in different clusters
$max\_diameter$ is the largest allowed distance between items in the same cluster

Question: Is it possible to partition I into k non-empty clusters so that the distance between any two items in different clusters is at least $min\_spacing$ and the distance between any two items in the same cluster is at most $max\_diameter$?

Solution (if exists): A surjective mapping $c\colon I \to \{1, 2, \ldots, k\}$ such that, for all $i, j \in I$,

$$\begin{aligned} D[i,j] &\geq min\_spacing && \text{if } c(i) \neq c(j); \\ D[i,j] &\leq max\_diameter && \text{if } c(i) = c(j). \end{aligned}$$

Can be verified in time $O(|I|^2)$.

From Clustering to Sat

- Reduce **Clustering** to **Sat** and solve it using a **Sat** solver:
  `https://tinyurl.com/4u7a925x`

Question: What if we had only one constraint instead of two (spacing and diameter)?
Is there a polynomial-time algorithm for one or both of the resulting problems?
Or maybe the original problem is already solvable in polynomial time?

Summary and Outlook

NP-complete problems are the hardest in NP.

Polynomial runs of NTMs can be described in propositional logic (Cook-Levin).

Any problem in NP can be solved by first reducing it to **Sat** and then using a **Sat**-solver. However, this may not always be the most efficient approach.

What's next?

- More examples of problems
- The limits of NP
- Space complexities