Lecture 6: The Cook–Levin Theorem

Aalto University
School of Science
Department of Computer Science

Spring 2019
Agenda

- Boolean satisfiability
- CNF formulas and Boolean functions
- The Cook–Levin theorem
NP-complete Problems

- **Last lecture:**
  - We saw that TMSAT is NP-complete
  - Definition tied directly to the definition of NP
  - Does not really tell us anything new about NP

- **This lecture:**
  - Prove that a problem called \textit{CNF-SAT} is NP-complete
  - First example of a \textit{natural} NP-complete problem
  - Starting point for further NP-completeness proofs
Cook–Levin Theorem

- One of the founding results of computational complexity
  - CNF-SAT is NP-complete
  - Named after Stephen Cook and Leonid Levin
  - Both independently proved the theorem around 1971

Stephen Cook

Leonid Levin
Boolean Formulas

- Boolean formula is built from the following primitives:
  - Variables $x_1, x_2, \ldots, x_n$
  - Operators AND ($\land$), OR ($\lor$), and NOT ($\neg$)
  - Example: $\varphi = (x_1 \land x_2) \lor (x_2 \land x_3) \lor (x_3 \land x_1)$

Definition (Boolean formulas, recursive definition)

The set of *Boolean formulas* over variables $x_1, x_2, \ldots, x_k$ is defined as follows:

- $x_i$ is a Boolean formula for any $i = 1, 2, \ldots, n$.
- If $\varphi$ is Boolean formula, then $\neg \varphi$ is Boolean formula.
- If $\varphi$ and $\psi$ are Boolean formulas, then $\varphi \land \psi$ and $\varphi \lor \psi$ are Boolean formulas.
Value of a Boolean Formula

- An assignment gives value 1 (true) or 0 (false) to each variable
  - Semantics of NOT, AND and OR are defined in the obvious way

Definition (Value of a Boolean formula)

Let $z = (z_1, z_2, \ldots, z_n) \in \{0, 1\}^n$ be an assignment. The value $\varphi(z)$ of formula $\varphi$ under assignment $z$ is defined as follows:

- If $\varphi = x_i$, when $\varphi(z) = z_i$.
- If $\varphi = \neg \psi$, then $\varphi(z) = 1 - \psi(z)$.
- If $\varphi = \psi_1 \land \psi_2$, then $\varphi(z) = 1$ if $\psi_1(z) = \psi_2(z) = 1$, and $\varphi(z) = 0$ otherwise.
- If $\varphi = \psi_1 \lor \psi_2$, then $\varphi(z) = 1$ if $\psi_1(z) = 1$ or $\psi_2(z) = 1$, and $\varphi(z) = 0$ otherwise.
Value of a Boolean Formula

- **Assignment** $z$ *satisfies* formula $\varphi$ if $\varphi(z) = 1$
  - A formula is *satisfiable* if there is a satisfying assignment
  - A formula is *unsatisfiable* otherwise

- **Examples:**
  - $\varphi = (x_1 \land x_2) \lor (x_2 \land x_3) \lor (x_3 \land x_1)$
    - $\varphi$ is satisfiable
  - $\psi = (x_1 \lor \neg x_2) \land \neg x_1 \land x_2$
    - $\psi$ is unsatisfiable
Conjunctive Normal Form

- A formula in **conjunctive normal form** is a formula that is an AND of ORs:
  - Example: \((x_1 \lor x_2) \land (x_2 \lor \neg x_3) \land (x_3 \lor \neg x_1)\)

- Formally:
  - A CNF formula is a formula of form
    
    \[
    \bigwedge_{i=1}^{m} \left( \bigvee_{j=1}^{k} \ell_{i,j} \right),
    \]
    
    where each \(\ell_{i,j}\) is either an \(x\) or \(\neg x\) for some variable \(x\)
  - Terms \(\ell_{i,j}\) are called **literals**
  - Terms \(\bigvee_{j=1}^{k} \ell_{i,j}\) are called **clauses**
CNF-SAT and \( k \)-SAT

**Definition (CNF-SAT)**
- **Instance:** A CNF formula \( \phi \).
- **Question:** Is \( \phi \) satisfiable?

**Definition (\( k \)-SAT)**
- **Instance:** A CNF formula \( \phi \) such that each clause in \( \phi \) has at most \( k \) literals.
- **Question:** Is \( \phi \) satisfiable?

- **2-SAT instance:** \((x_1 \lor x_2) \land (x_2 \lor \lnot x_3) \land (x_3 \lor \lnot x_1)\)
- **3-SAT instance:** \((x_1 \lor x_2 \lor x_3) \land (x_2 \lor \lnot x_3 \lor x_4)\)
- **4-SAT instance:** \((x_1 \lor x_2 \lor x_3 \lor \lnot x_4) \land (x_2 \lor \lnot x_3 \lor x_4)\)
Theorem

*CNF-SAT is in NP.*

**Proof:** CNF-SAT has a polynomial-time verifier

- **Input:** a formula $\varphi$ over variables $x_1, x_2, \ldots, x_n$
- **Certificate:** an assignment $z \in \{0, 1\}^n$
- **Verification algorithm:** compute the value $\varphi(z)$, accept if $\varphi(z) = 1$

Corollary

*For any fixed $k \geq 1$, $k$-SAT is in NP.*
Universality of CNF Formulas

- CNF formulas can express all Boolean functions
  - May require exponential number of clauses
  - This does not matter: we want to use this construction for \textit{constant} number of variables

Lemma

Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be a Boolean function. Then there is a CNF formula $\varphi$ over $n$ variables with at most $2^n$ clauses such that $\varphi(z) = f(z)$ for all $z \in \{0, 1\}^n$. 
Universality of CNF Formulas: Proof

- For each \( z \in \{0, 1\}^n \), we construct clause \( C_z \):
  - Let \( \ell_i = x_i \) if \( z_i = 0 \), and \( \ell_i = \neg x_i \) if \( z_i = 1 \)
  - Let \( C_z = \bigvee_{i=1}^n \ell_i \)
  - We now have \( C_z(y) = 0 \) if \( z = y \), and \( C_z(y) = 1 \) if \( z \neq y \)

- For any \( f : \{0, 1\}^n \rightarrow \{0, 1\} \), we construct formula \( \varphi \):
  - Let \( \varphi_f = \bigwedge_{z: f(z) = 0} C_z \)
  - If \( f(y) = 0 \), then \( y \) does not satisfy the clause \( C_y \) in \( \varphi_f \)
  - If \( f(y) = 1 \), then \( y \) satisfies all clauses \( C_y \) in \( \varphi_f \)
  - Thus, we have \( \varphi_f(y) = f(y) \) for all \( y \in \{0, 1\}^n \)
Cook–Levin Theorem

**Theorem**

*CNF-SAT is NP-complete.*

- **We have:** CNF-SAT is in NP
- **Next:** CNF-SAT is NP-hard
Cook–Levin Theorem: Proof

- **General template for the proof:**
  - Let $L \in \text{NP}$ be a language
  - We prove that there is a polynomial-time reduction from $L$ to CNF-SAT

- **The only thing we know about $L$ is that it is in NP**
  - There exists a verifier $M$ for $L$
  - For any $x \in L$, there is a certificate for $x$ of length at most $q(|x|)$, for some polynomial $q$
  - $M$ runs on input $(x, u)$ in time $p(|x|)$ for some polynomial $p$ with $q(n) \leq p(n)$
  - $M$ uses at most $p(|x|)$ positions on each tape
  - We may assume $M$ has one working tape, uses alphabet $\{\triangleright, \Box, 0, 1\}$
Execution Tables

- **Execution of** $M$ **on input** $(x, u)$ **can be viewed as a table:**
  - Row $i$ describes the state of $M$, the positions of heads and the contents of the tapes after step $i$
  - Since $M$ runs in time $p(|x|)$, each row needs to store at most $1 + 3 \cdot 2 \cdot p(|x|)$ entries
    - three tapes, one (head,symbol)-pair per position on a tape
  - The number of rows is at most $p(|x|)$, and wlog we may assume exactly $p(|x|)$ (no moves after $M$ enters halting state)

- **Table can be encoded as binary:**
  - $|Q|$ bits for state
  - 3 bits per each tape position on each of 3 tapes
    - 1 bit for head marker (location indicator)
    - 2 bits for current symbol encoding
Execution Tables

- **Execution table is accepting if:**
  - State entry on the last row corresponds to the halting state
  - The encoding of the output tape on the last row corresponds to ▶1□□□... ▶

- **By definition:**
  - $M$ has accepting execution table if and only if $M$ accepts
**Proof overview:**

- Let $x$ be an instance of $L$.
- We construct a CNF-SAT formula $\varphi_x$ over $S = p(|x|) \cdot (|Q| + 9p(|x|))$ variables.
- Assignment $z$ to $\varphi_x$ encodes an execution table of $M$.
- Formula $\varphi_x$ is construed so that a given assignment $z$ satisfies $\varphi_x$ if and only if:
  
  (i) $z$ encodes a valid execution table.
  (ii) $z$ encodes an execution table on input $(x, u)$ for some $u \in \{0, 1\}^*$.
  (iii) $z$ encodes an accepting execution table.
Cook–Levin Theorem: Proof

- Clauses of $\varphi_x$ are constructed to \textit{locally} enforce the constraints
  - We could use the universality lemma to directly construct a CNF formula to enforce that the variables encode an accepting execution table
  - This would give \textit{exponential} size in terms of $|x|$  
  - Need to be more careful to get polynomial size

- Basic idea: encode \textit{local} constraints
Cook–Levin Theorem: Proof

- **Clauses of $\varphi_x$ that enforce the starting and halting conditions:**
  - Contents of the input tape on the first row of the execution table is $\triangleright x \square \square \ldots$ and of the other tapes $\triangleright \square \square \ldots$
  - All heads start at position 1 and the first state is $q_0$
  - State on the last row of the execution table is $q_h$
  - Contents of the output tape on the last row of the execution table is $\triangleright 1 \square \square \ldots$

- These can be encoded by a conjunction of $O(p(|x|))$ single-literal clauses
Cook–Levin Theorem: Proof

- **Clauses of $\varphi_x$ that enforce consistency of the table:**
  - Only single head position and state for each row
  - If head is not at position $j$, then the tape symbols at position $j$ do not change between steps
  - If head is at position $j$, then the tape symbols and the head markers around position $j$ change correctly between steps
  - The machine state changes correctly between steps

- **Each of these conditions can be viewed as a Boolean function on a constant number of variables**
  - At most $c = 2|Q| + 3 \cdot 2 \cdot 6$ variables per constraint
  - Encode as a CNF with $2^c$ clauses using universality lemma
  - About $O\left(p(|x|)^2\right)$ constraints needed
Cook–Levin Theorem: Proof

The final CNF formula $\varphi_x$ is conjunction of all constraints:
- A conjunction of CNF formulas is a CNF formula

By construction, this gives us a reduction from $L$ to CNF-SAT
- $x \in L$ if and only if $\varphi_x$ is satisfiable
- $\varphi_x$ can be constructed in polynomial time
CNF-SAT: Discussion

- **CNF-SAT is a relevant problem in practice**
  - *Problem-specific* reductions to CNF-SAT can be much more compact than the general reduction given by Cook–Levin theorem
  - E.g. Intel has used CNF-SAT solvers to verify and optimise processor designs
  - Highly efficient CNF-SAT solvers are available as open-source software
  - For many difficult optimisation problems, reducing to CNF-SAT and applying off-the-shelf solvers can be much faster than anything you implement yourself
Lecture 6: Summary

- CNF-SAT and $k$-SAT
- CNF-SAT is NP-complete