Efficient Algorithms and Problem Complexity

– Reductions, NP-completeness, and the Cook-Levin Theorem –

Frank Drewes
Department of Computing Science
Umeå University
Today’s Menu

1. Reductions

2. NP-Completeness

3. The Cook-Levin Theorem
What are We Aiming at?

- We suspect some problems in NP to be significantly harder than each of the problems in P.
- How should we define significance (or insignificance) in this context?
- The definition should consider
  1. the difference between problems in P to be insignificant, but
  2. the difference between problems in and outside P to be significant.
- The Composition Theorem gives us a strong hint: We can apply any polynomial-time preprocessing to a problem in P without leaving P!
- The preprocessing may in particular translate one problem into another.
- Such translations are called polynomial-time reductions.
Polyomial-Time Reductions

Definition (polynomial-time reduction)
Let $A$ and $B$ be decision problems. A polynomial-time reduction from $A$ to $B$ is a function $f$ such that

1. for every input $x$, $x \in A \iff f(x) \in B$, and
2. $f$ can be computed in (deterministic!) polynomial time.

We say that $A$ is polynomial-time reducible to $B$, and write $A \leq_p B$.

Note: $\leq_p$ is a preorder (also called a quasi-order), that is,

- it is transitive ($A \leq_p B \leq_p C$ implies $A \leq_p C$) [WHY?]
- it is reflexive ($A \leq_p A$) [WHY?].
So, What Does this Intuitively Mean?

Importantly, $\leq_p$ is not acyclic:

For $A \neq B$ we may have both $A \leq_p B$ and $B \leq_p A$. We write this as $A \equiv_p B$ and say that $A$ and $B$ are polynomial-time equivalent.

- $A \leq_p B$ formalizes that $A$ is not significantly harder than $B$ – the difference is only a polynomial preprocessing.
- In particular, $A \equiv_p B$ means that the difference is only polynomial in either direction – they are of the same complexity if we disregard polynomial preprocessing.
- On the other hand, if $A \leq_p B$ but $B \not\leq_p A$, then $B$ is indeed significantly harder than $A$, i.e., the difference is more than “just” a polynomial.
Insignificance of the Difference Between Problems in P

Now, is “the difference between problems in P insignificant”?

We want that a polynomial-time reduction from $A$ to $B$ exists if $A, B \in P$.

Here is such a reduction $f$: Choose any $y^+ \in B$ and $y^- \notin B$, and define

$$f(x) = \begin{cases} y^+ & \text{if } x \in A \\ y^- & \text{otherwise.} \end{cases}$$

- **Checking requirement 1**: by the definition of $f$, $x \in A \iff f(x) \in B$.
- **Checking requirement 2**: computing $f$ in polynomial time works by deciding whether $x \in A$ and outputting either $y^+$ or $y^-$. 
- Intuitively, the reduction does all the work itself.
- It can do so, because it has all the resources $A$ requires.
- There are exactly two problems $B$ for which this does not work. Which ones?
Significance of the Difference Between Problems In and Outside P

Theorem (P is (backwards) closed under reductions)

If \( A \leq_p B \) and \( B \in P \) then \( A \in P \).
In other words, if \( B \in P \) but \( A \not\in P \) then \( A \not\leq_p B \).

Proof: Let \( M \) decide \( B \) and let \( M_0 \) compute a reduction \( f \) from \( A \) to \( B \), both in polynomial time.

- \( M' = M \circ M_0 \) runs in polynomial time (Composition Theorem).
- We have \( M'(x) = M(M_0(x)) = M(f(x)) \). Therefore,
  \[
  x \in A \iff f(x) \in B \iff M(f(x)) = 1 \iff M'(x) = 1.
  \]
- Thus, \( M' \) decides \( A \) in polynomial time.
NP-Completeness

We ultimately want to understand whether there are problems in NP that are not in P. So, it makes sense to look at the hardest problems in NP.

**Definition (NP-complete)**

A decision problem $B$ is NP-complete if

- $B ∈ \text{NP}$ and
- $A \leq_p B$ for every problem $A ∈ \text{NP}$.

- If the first condition is dropped, we say that $B$ is NP-hard.
- For classes above NP (such as EXP), completeness is defined similarly.
- For P and classes inside P, we would need another type of reduction (because of the reasoning on Slide 6).
To prove $P = NP$, it suffices to solve one NP-complete problem efficiently.

**Theorem**

If there is an NP-complete problem $B$ such that $B \in P$, then $P = NP$.

**Proof:** Since $B$ is NP-complete, we have $A \leq_p B$ for all $A \in NP$. By the closedness of $P$ under reductions (Slide 7), this means that $A \in P$. 
Proving NP-Completeness

Lemma

If $A \in \text{NP}$ and $B \leq_p A$ for an NP-complete problem $B$, then $A$ is NP-complete.

Proof: This is because, for every $C \in \text{NP}$, we have $C \leq_p B \leq_p A$.

This yields the most common way to prove that $A$ is NP-complete:

1. show that $A \in \text{NP}$,
2. choose a suitable NP-complete problem $B$, so that you manage to find a reduction $f$ from $B$ to $A$ (note the direction!!),
3. argue that $f$ is polynomial-time computable,
4. show that $f(x) \in A$ iff $x \in B$ (both directions!!!).
The Cook-Levin Theorem

For the method on the previous slide to be useful, someone has to find a first NP-complete problem...

Theorem (Cook 1971, Levin 1973)

SAT is NP-complete.

The proof...

- shows how to construct a polynomial-time reduction from $A$ to SAT, for arbitrary $A \in \text{NP}$,
- cannot be based on anything else than the existence of a nondeterministic polynomial-time decision algorithm for $A$.

Rather than nRAMs, we use Turing machines (TMs) for the proof sketch, because they are simpler.
The Cook-Levin Theorem

Proof Sketch of the Cook-Levin Theorem

We use the characterization of NP by polynomially bounded binary relations. Let $M$ be a deterministic TM and $p, q$ polynomials such that

- $M$ accepts only strings of the form $x \# y$, where $y \in \{0, 1\}^{p(n)}$ ($n = |x|$),
- $x \in A$ if and only if $\exists y \in \{0, 1\}^{p(n)}$ with $M(x \# y) = yes$, and
- $M$ runs in at most $q(n)$ steps.
- When $M$ halts (after $q(n)$ steps), its head scans the first tape cell.
- $M$ accepts the input by writing a special symbol $\alpha$ in the first tape cell and entering a special accepting state $z_{yes}$.

**Major proof step:** From $x \# y$ we construct, in polynomial time, a Boolean formula (in CNF) that encodes the entire computation of $M$ with this input.
A computation of $M$ as a $q(n) \times q(n)$-array of symbols:

$$q(n)$$

<table>
<thead>
<tr>
<th>$a^{z_0}$</th>
<th>$b$</th>
<th>$b$</th>
<th>$\cdots$</th>
<th>$#$</th>
<th>0</th>
<th>1</th>
<th>$\cdots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c$</td>
<td>$b^{z_1}$</td>
<td>$b$</td>
<td>$\cdots$</td>
<td>$#$</td>
<td>0</td>
<td>1</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$\cdots$</td>
<td>$b$</td>
<td>$a^{z_k}$</td>
<td>$c$</td>
<td>$\cdots$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\cdots$</td>
<td>$b$</td>
<td>$b$</td>
<td>$c^{z_l}$</td>
<td>$\cdots$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$\alpha^{z_{yes}}$ $\cdots$

- Encode contents of cell $c_{ij}$ by a vector $\vec{c}_{ij}$ of boolean variables.
- Row 1 should contain the initial configuration
  $\Rightarrow$ use formulas $\varphi_{\sigma}(j) \equiv " \vec{c}_{1j} \text{ represents } \sigma \"$.
- $c_{i+1j}$ is determined by $c_{ij-1}, c_{ij}, c_{ij+1}$
  $\Rightarrow$ use $\varphi(i, j) \equiv " \vec{c}_{i+1j} \text{ represents the successor of } \vec{c}_{ij-1}, \vec{c}_{ij}, \vec{c}_{ij+1} \"$.
- $c_{q(n)1}$ should indicate acceptance
  $\Rightarrow$ use $\varphi_{yes} \equiv " \vec{c}_{q(n)1} \text{ represents } \alpha^{z_{yes}} \".$
The Cook-Levin Theorem

Proof Sketch of the Cook-Levin Theorem

Computing the reduction for input $x \# y$ of length $n$:

1. Compute $q(n)$ and the initial configuration $b_1 \cdots b_{q(n)}$.
2. Print “$\varphi_{b_1}(1), \ldots, \varphi_{b_{q(n)}}(q(n))$”.
3. For $i = 1, \ldots, q(n) - 1$ and $j = 1, \ldots, q(n)$ print “$\land \varphi(i, j)$”.
4. Print “$\land \varphi_{yes}$”.

$\Rightarrow$ the resulting formula $\varphi$ is satisfiable if and only if $M$ accepts $x \# y$.

But we have only $x$ and want $\varphi$ to be satisfiable if there exists such a $y$!

Easy solution: In Step 2 above, print $\varphi_{b_j}(j)$ only for the initial part $x \#$. For the bits in $y$, print $\varphi_0(j) \lor \varphi_1(j)$ (turned into CNF).
Please read the corresponding part of the lecture notes. The proof sketch of the Cook-Levin Theorem differs from the one above in some details to make it more exact but is essentially the same.