Efficient Algorithms and Problem Complexity
– Techniques for Constructing Reductions –

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

1. Different Types of Reductions
2. Reduction by Restriction
3. Reduction by Local Replacement
4. Reduction by Composition of Gadgets
Types of Reductions

Notation: NPC denotes the class of all NP-complete problems.

Suppose we want to show that $A \in \text{NPC}$. If we already know that $A \in \text{NP}$, we have to find a problem $B \in \text{NPC}$ and a polynomial-time reduction from $B$ to $A$.

The three most common types of reductions:

1. reduction by restriction (simple)
2. reduction by local replacement (still usually rather simple)
3. reduction by composition of “gadgets” (can be quite tricky)
Reduction by Restriction

Idea: If $A$ is a more general variant of a problem $B \in \text{NPC}$, then the reduction only has to turn an instance of $B$ into an instance of $A$.

Intuition: If $A$ is more general than $B$, then $A$ cannot be easier than $B$. 
Example: Reducing HAM to dHAM

Recall Hamiltonian Cycle (HAM)

**Input:** An undirected graph $G = (V, E)$.

**Question:** Does $G$ contain a simple cycle of length $|V|$?

Directed Hamiltonian Cycle (dHAM) is defined in the same way, except that $G$ is directed (and a directed simple cycle of length $|V|$ is sought).

Assume that we already know that $HAM \in NPC$.

In which sense is dHAM a more general variant of HAM?
**Example: Reducing HAM to dHAM**

Reduction $f$ from HAM to dHAM: In the undirected input graph $G$, turn every edge $\bullet\longrightarrow\bullet$ into two antiparallel edges: $\bullet\longleftarrow\longrightarrow\bullet$.

Correctness:

- Computability of $f$ in polynomial time is obvious.
- If $G \in \text{HAM}$, then it has a Hamiltonian cycle $v_0 \cdots v_n$. Thus, $(v_{i-1}, v_i)$ is a directed edge in $f(G)$ for all $i \in \{1, \ldots, n\}$, which means that $v_0 \cdots v_n$ is a directed cycle in $f(G)$, i.e., $f(G) \in \text{dHAM}$.
- If $f(G) \in \text{dHAM}$, then it has a directed Hamiltonian cycle $v_0 \cdots v_n$. Since $f(G)$ contains an directed edge $(v_{i-1}, v_i)$ only if $G$ contains the corresponding undirected edge, this means that $v_0 \cdots v_n$ is a Hamiltonian cycle in $G$. In other words, $G \in \text{HAM}$.
Example: Reducing HAM to TSP\(_D\)

One version of the Travelling Salesman Decision Problem (TSP\(_D\))

**Input:** An \( n \times n \)-matrix of distances \( d_{i,j} \in \mathbb{N} \) and a number \( k \in \mathbb{N} \).

**Question:** Is there a tour \( v_0, \ldots, v_{n-1} \) such that 
\[
\{ v_0, \ldots, v_{n-1} \} = \{ 1, \ldots, n \} \text{ and } \sum_{j=1}^{n} d_{v_{j-1}, v_{j \mod n}} \leq k
\]

Assume again that we already know that HAM \( \in \text{NPC} \).

**In which sense is TSP\(_D\) a more general variant of HAM?**
**Example: Reducing HAM to TSP\(_D\)**

Reduction \( f \) from HAM to TSP\(_D\): Let \( V = \{1, \ldots, n\} \) be the set of nodes of the input graph \( G \). Let \( k = n \) and, for \( i, j \in \{1, \ldots, n\} \),

\[
    d_{i,j} = \begin{cases} 
        1 & \text{if } G \text{ contains the edge } (i, j) \\
        2 & \text{otherwise.}
    \end{cases}
\]

Correctness:

- Computability of \( f \) in polynomial time is again obvious.
- If \( G \in \text{HAM} \), then it has a Hamiltonian cycle \( v_0 \cdots v_n \). Thus, \( d_{v_{j-1}, v_{j \mod n}} = 1 \) for all \( i, j \in \{1, \ldots, n\} \), which means that the tour \( v_0 \cdots v_{n-1} \) has length \( n = k \).
- If \( f(G) \in \text{TSP}_D \), then there is a tour \( v_0 \cdots v_{n-1} \) of length \( \leq n \). Since the only distances are 1 and 2, this means that all the distances on this tour are 1. Thus, \( v_0 \cdots v_{n-1}v_0 \) is a Hamiltonian cycle in \( G \).
Idea: To turn an instance of $B \in \text{NPC}$ into a corresponding instance of $A$, we locally replace substructures of an instance of $B$ by other substructures.

This is often used in order to turn a more general problem into a special form, showing that even this special form is NP-complete.
Example: Reducing SAT to 3SAT

Recall (?) 3-Satisfiability (3SAT)

**Input:** An propositional formula \( \varphi \) in CNF in which each clause has exactly 3 literals.

**Question:** Is \( \varphi \) satisfiable?

We already know that \( \text{SAT} \in \text{NPC} \).

How can we turn an instance of \( \text{SAT} \) into an equivalent instance of \( 3\text{SAT} \)?

We replace every clause by a bunch of clauses consisting of 3 literals each.
Example: Reducing SAT to 3SAT

Replacing a clause \((l_1 \lor \cdots \lor l_k)\) by clauses consisting of 3 literals each, using new variables \(y_i\):

\[
\begin{align*}
(l) & \quad \mapsto \quad (l \lor y_1 \lor y_2) \land \\
& \quad \quad (l \lor y_1 \lor \neg y_2) \land \\
& \quad \quad (l \lor \neg y_1 \lor y_2) \land \\
& \quad \quad (l \lor \neg y_1 \lor \neg y_2)
\end{align*}
\]

\[
\begin{align*}
(l_1 \lor l_2) & \quad \mapsto \quad (l_1 \lor l_2 \lor y_1) \land (l_1 \lor l_2 \lor \neg y_1)
\end{align*}
\]

\[
\begin{align*}
(l_1 \lor \cdots \lor l_k) & \quad \mapsto \quad (l_1 \lor l_2 \lor y_1) \land \\
& \quad \quad (\neg y_1 \lor l_3 \lor y_2) \land \\
& \quad \quad (\neg y_2 \lor l_4 \lor y_3) \land \\
& \quad \quad \vdots \\
& \quad \quad (\neg y_{k-4} \lor l_{k-2} \lor y_{k-3}) \land \\
& \quad \quad (\neg y_{k-3} \lor l_{k-1} \lor l_k)
\end{align*}
\]
Example: Reducing SAT to 3SAT

Correctness of \( C = (l_1 \lor \cdots \lor l_k) \mapsto (l_1 \lor l_2 \lor y_1) \land (\neg y_1 \lor l_3 \lor y_2) \land \cdots \land (\neg y_{k-4} \lor l_{k-2} \lor y_{k-3}) \land (\neg y_{k-3} \lor l_{k-1} \lor l_k) = C' \)

- \( \alpha(C') = true \) for an assignment \( \alpha \)
  \( \Rightarrow \) \( \alpha(l_i) = true \) for some \( i \)
  \( \Rightarrow \) extending \( \alpha \) by \( \alpha(y_1) = \cdots = \alpha(y_{i-2}) = true \) and \( \alpha(y_{i-1}) = \cdots = \alpha(y_{k-3}) = false \) yields \( \alpha(C') = true \).

- \( \alpha(C'') = true \) for an assignment \( \alpha \)
  Consider the first clause that does not contain a \( y_i \) with \( \alpha(y_i) = true \)
  \( \Rightarrow \) this clause does not contain \( \neg y_{i-1} \) with \( \alpha(\neg y_{i-1}) = true \) either
  \( \Rightarrow \) the clause contains \( l_j \) with \( \alpha(l_j) = true \)
  \( \Rightarrow \) \( \alpha(C') = true \).
Gadgets are often used if the target of the reduction is a graph problem. **Idea:** Given an instance of $B \in \text{NPC}$ we build a corresponding instance of $A$ by composing copies of one or more “gadgets”. The gadgets are building blocks constructed to fulfill a specific purpose. **Intuition:** Think of composing, i.e., a binary adder using logical gates, or composing an 8-bit adder from 7 binary adders.
Example: Reducing SAT to dHAMPATH

Recall (?) directed Hamiltonian Path (dHAMPATH)

Input: A directed graph $G = (V, E)$.

Question: Does $G$ contain a simple path of length $|V| - 1$?

We want to reduce SAT to dHAMPATH.

How can we turn an instance of SAT into an equivalent instance of dHAMPATH?

Consider a formula $\varphi$ in CNF with $n$ variables $x_1, \ldots, x_n$. Let $l_2, l_4, \ldots, l_m$ be the sequence of literals in (the clauses of) $\varphi$. 

A gadget for choosing the truth value of $x_i$:

$$\alpha(x_i) = \text{true}$$

Notes:

- A path will enter the gadget at $v^i_1$ and leave it at $v^i_m$ if $\alpha(x_i) = \text{true}$ and conversely if $\alpha(x_i) = \text{false}$.
- The gadget contains twice as many nodes as $\varphi$ contains literals.
- Another type of gadget will be used for the clauses.
- If a clause contains the literal $l_j \in \{x_i, \neg x_i\}$, its gadget will be attached to $v^i_{j-1}$ and $v^i_j$. 
The gadget for the clauses:

\[ \ldots x_i \ldots \text{ or } \ldots \neg x_i \ldots \] (clause with literal \( l_j \))

Notes:

- One extra node per clause \( C \).
- If \( l_j = x_i \) or \( l_j = \neg x_i \) is in \( C \), it is connected to \( v_{j-1}^i, v_j^i \) as shown.
- Passing the \( x_i \)-gadget left to right, we can make a “detour” to pass \( C \) if the literal \( l_j = x_i \) occurs in \( C \).
- Similarly when passing from right to left and \( l_j = \neg x_i \) occurs in \( C \).
Putting it all together:

\[ C_1 \xrightarrow{\ldots} \ldots \ldots \ldots \xrightarrow{\ldots} C_k \]

\[ x_1 \text{-gadget} \]

\[ x_2 \text{-gadget} \]

\[ \ldots \]

\[ x_n \text{-gadget} \]

\[ \text{start} \rightarrow \text{end} \]
Correctness, direction 1: Suppose $\alpha$ makes $\varphi$ true.

- Start at $\text{start}$. 
- Continue to $v_1^1$ if $\alpha(x_1) = \text{true}$; otherwise, go to $v_m^1$.
- Pass through the $x_1$-gadget with detours via clause gadgets (see below).
- Continue similarly with the $x_2$-gadget, and so on.
- Detours when passing the $x_i$-gasket from left to right ($\alpha(x_i) = \text{true}$): Go from $v_{j-1}^i$ to $v_j^i$ via $C$ (rather than directly from $v_{j-1}^i$ to $v_j^i$) if
  - $C$ contains the literal $l_j = x_i$ and
  - (the node corresponding to) $C$ had not been passed earlier.
- Similarly if $\alpha(x_i) = \text{false}$ and $C$ contains the literal $l_j = \neg x_i$.

As $\alpha(C) = \text{true}$ for every clause $C$, this visits each node once.
Example: Reducing SAT to dHAMPATH

Correctness, direction 2: Suppose $f(\varphi) \in \text{dHAMPATH}$.

- The path must start at start, pass the $x_i$-gadgets one after another, and end at end.
- In particular, if a clause node is entered via $v^i_{j-1}$ it must be left via $v^i_j$, and vice versa (otherwise, nodes are left out or used twice).
- Depending on the direction in which the $x_i$-gadgets are passed, this yields a truth assignment $\alpha$.
- Since all the clause nodes are on the paths, they must have been included via detours.
- A detour is only possible if the corresponding literal is made true by $\alpha$.

Hence, $\alpha(C) = \text{true}$ for all clauses $C$, meaning that $\alpha(\varphi) = \text{true}$. 
Please read and understand the NP-completeness proofs in any of the two textbooks (or in another book).