Computationally Hard Problems

Proofs of $\mathcal{NP}$-Completeness

Carsten Witt

Institut for Matematik og Computer Science
Danmarks Tekniske Universitet

Fall 2020

Steps for proving $\mathcal{NP}$-completeness (rep.)

We will now show for further problems that they are $\mathcal{NP}$-complete, i. e., presumably hard to solve.

- Prove that $P \in \mathcal{NP}$.
- Find a suitable problem P_c which is known to be $\mathcal{NP}$-complete.
- Prove $P_c \leq_p P$.
  (1) Describe a transformation T which transforms every instance $\boldsymbol{X}$ of P_c into an instance $T(\boldsymbol{X})$ of P and which runs polynomial in the size $\|\boldsymbol{X}\|$ of $\boldsymbol{X}$.
  (2) Show that if the answer to $\boldsymbol{X}$ is YES then so is the answer to $T(\boldsymbol{X})$.
  (3) Show that if the answer to $T(\boldsymbol{X})$ is YES then so is the answer to $\boldsymbol{X}$.
  (Alternative: if the answer to $\boldsymbol{X}$ is NO then so is the answer to $T(\boldsymbol{X})$.)

Overview

We know that SAT and 3-SAT are $\mathcal{NP}$-complete.

We will now establish $\mathcal{NP}$-completeness for quite different problems.

The problems will be generic problems for different types of problems:

- Satisfying boolean formulas
- Optimal selections
- Partitioning
- Covering
- Tours/cycles

Techniques for Polynomial-Time Reductions (1/2)

Proving a problem P $\mathcal{NP}$-complete typically involves a pol.-time reduction from a reference problem P_c, i. e., $P_c \leq_p P$.

Such reductions can informally be categorized according to the techniques used. In increasing order of difficulty, we have:

Modeling Show that P is exactly the same as P_c if P is expressed in sufficiently abstract terms.

Restriction Show that P contains P_c as a special case.

Local Replacement Introduce for every single component of P_c one or several components of P.

Component Design Introduce new components that integrate different components of P_c into a single component of P.

Techniques for Polynomial-Time Reductions (2/2)

Examples for the different techniques:

Modeling $\textsc{SubsetSum} \leq_p \textsc{PotatoSoup}$

Restriction $\text{3-SAT} \leq_p \text{SAT}$,
$\textsc{Partition} \leq_p \textsc{SubsetSum}$

Local Replacement $\text{SAT} \leq_p \text{3-SAT}$
(every clause is replaced by a number of clauses)

Component Design $\text{3-SAT} \leq_p \textsc{SubsetSum}$,
$\text{3-SAT} \leq_p \textsc{VertexCover}$,
$\text{3-SAT} \leq_p \textsc{DirectedHamiltonCircuit}$
$\textsc{SubsetSum} \leq_p \textsc{Partition}$ (all to come)

SubsetSum

Problem [SubsetSum]
Input: Natural numbers $s_1, \ldots, s_n$ and B.
Output: YES if there is a set $A \subseteq \{1, \ldots, n\}$ such that

$$\sum_{i \in A} s_i = B \ , \tag{1}$$

and NO otherwise.

SubsetSum is in $\mathcal{NP}$

Theorem

The problem SubsetSum *is $\mathcal{NP}$-complete.*

It is now an "easy exercise" to prove that is in $\mathcal{NP}$.

- A random string of bits is interpreted as a subset $A \subseteq \{1, \ldots, n\}$,
- check whether $\sum_{i \in A} s_i = B$.
- If so answer YES, and NO otherwise.
- If true answer is YES, there is a positive probability of the random string encoding it, making the algorithm saying YES.
- If true answer is NO, then no subset A with $\sum_{i \in A} s_i = B$ exists and the algorithm must say NO.
- The whole procedure runs in time polynomial in the input size.

SubsetSum is $\mathcal{NP}$-complete

The reference problem to reduce from is 3-Sat.

$$\text{3-Sat} \leq_p \text{SubsetSum}$$

We show how to transform 3-Sat instances into SubsetSum instances.

This is done by defining (huge) numbers which are "related" to the clauses of 3-Sat.

The numbers reserve a digit for each clause and each variable.

A satisfying assignment will correspond to a certain way of adding certain numbers up with result B.

This technique is called *(connected) component design*.

The Instance

The 3-SAT INSTANCE:

- $X := \{x_1, x_2, \ldots, x_n\}$ – the boolean variables
- $C := \{c_1, c_2, \ldots, c_m\}$ – the set of 3-clauses

The SUBSETSUM instance

- Numbers a_i, b_i, d_j, e_j and B with $n + m$ decimal digits each
- $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$
- Altogether $2n + 2m + 1$ numbers
- The last n digits are for variables
- and the first m for clauses.
- The a_i and b_i encode the main structure, the d_j and e_j are "fill-up" elements

The Instance: Last n Digits

First, let us look only at the last n digits:

a_i and b_i have a 1 at the i-th of these, rest 0.

d_j and e_j contain only 0-entries.

Example for $n = 4$:

$$
\begin{aligned}
a_1 &= \ldots 1000 \\
a_2 &= \ldots 0100 \\
a_3 &= \ldots 0010 \\
a_4 &= \ldots 0001 \\
b_1 &= \ldots 1000 \\
b_2 &= \ldots 0100 \\
b_3 &= \ldots 0010 \\
b_4 &= \ldots 0001
\end{aligned}
$$

The last n digits of B are all $\ldots 1111$.

So must choose either a_i or b_i to form last digits of B.

Interpretation: Choosing a_i means $x_i = \text{true}$, choosing b_i means $x_i = \text{false}$.

The Instance: First m Digits

The first m digits of the a_i and b_i state in which clauses x_i and $\bar{x}_i$ appear, respectively.

Let $c_j = z_1 \vee z_2 \vee z_3$ be the j-th clause.

If $z_1 = x_k$ then a_k gets a 1 at the j-th digit (from left). If $z_1 = \bar{x}_\ell$ then b_ℓ gets a 1 at the j-th digit. And so on for z_2 and z_3.

Example $c_1 = x_1 \vee \bar{x}_2 \vee x_3$:

$$\begin{array}{ll} a_1 = 1\ldots & b_1 = 0\ldots \\ a_2 = 0\ldots & b_2 = 1\ldots \\ a_3 = 1\ldots & b_3 = 0\ldots \\ a_4 = 0\ldots & b_4 = 0\ldots \end{array}$$

The first m digits of B are $333\ldots$, hence $B = \underbrace{3\ldots3}_{m}\underbrace{1\ldots1}_{n}$

Consequence: If we choose the a_i and b_i according to an assignment satisfying c_j, the j-th digit from left adds up to 1, 2, or 3.

The Instance: Fill-up Elements

So far: if we have a satisfying assignment for c_j then choosing the a_i or b_i accordingly yields *at least* 1 in the j-th sum digit.

Problem: Only if all three literals evaluate to true, we get at 3 there. But that is not guaranteed for all satisfiable instances.

To fill up, let d_j and e_j have a 1 in the j-th digit from left, rest 0.

This way, we can increase the digit by 1 or 2 (but not 3).

Example: $c_1 = x_1 \vee \overline{x}_2 \vee x_3$, $c_2 = \overline{x}_1 \vee x_2 \vee \overline{x}_4$ and $c_3 = \overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3$.

	c_1	c_2	c_3	x_1	x_2	x_3	x_4
a_1	1	0	0	1	0	0	0
a_2	0	1	0	0	1	0	0
a_3	1	0	0	0	0	1	0
a_4	0	0	0	0	0	0	1
b_1	0	1	1	1	0	0	0
b_2	1	0	1	0	1	0	0
b_3	0	0	1	0	0	1	0
b_4	0	1	0	0	0	0	1
d_1	1	0	0	0	0	0	0
d_2	0	1	0	0	0	0	0
d_3	0	0	1	0	0	0	0
e_1	1	0	0	0	0	0	0
e_2	0	1	0	0	0	0	0
e_3	0	0	1	0	0	0	0
B	3	3	3	1	1	1	1

Correctness of the Reduction: "$\Rightarrow$"

Construction is doable in time $O((n+m)^2)$, i. e., polynomial in input size (even though we create *huge* numbers).

If instance to 3-SAT is YES-instance then satisfying assignment translates to a choice for

- a subset of the a_i, b_i (according to whether x_i is true or false)
- and a subset of the d_j and e_j (according to whether 1, 2, or 3 literals are true in c_j)
- with total sum B.

Hence, we have a YES-instance to SUBSETSUM.

Correctness of the Reduction "$\Leftarrow$"

If instance to SUBSETSUM is YES-instance then there is a subset summing up to $3 \dots 31 \dots 1$.

In each digit, at most 5 numbers have a 1-entry: more precisely, there are 2 such numbers for the last n digits by construction, and 5 for the first m since each clause has 3 literals and there are two fill-up elements. Therefore, no carry-over is possible.

Hence, to get the 1s in second part, either a_i or b_i is chosen, for every $i \in \{1, \dots, n\}$. We get a proper truth assignment.

We cannot reach a 3 at the j-th digit, $j \in \{1, \dots, m\}$, just using d_j and e_j. Hence, one of the a_i or b_i must be chosen for the digit, which corresponds by construction of the truth assignm. to satisfying clause j. Hence, every clause is satisfied.

Hence, the instance to 3-SAT is a YES-instance, too.

Altogether

Altogether: 3-SAT $\leq_p$ SUBSETSUM PROVED.

Partition

Problem [Partition]
Input: Natural numbers $s_1, \ldots, s_n$.
Output: YES if there is a set $A \subseteq \{1, \ldots, n\}$ such that

$$\sum_{i \in A} s_i = \sum_{i \in \{1,\ldots,n\} \setminus A} s_i \ , \tag{2}$$

and NO otherwise.

Theorem

The problem Partition *is $\mathcal{NP}$-complete.*

Partition is in $\mathcal{NP}$

Works very in a similar way as with SubsetSum.

PARTITION is $\mathcal{NP}$-complete

We design a reduction from similarly looking SUBSETSUM.

$$\text{SUBSETSUM} \leq_p \text{PARTITION}$$

Note that PARTITION is special case of SUBSETSUM, hence the opposite reduction is trivial.

The required reduction is again by component design, with only two additional components.

The Instance

Let $(s_1, \ldots, s_n, B)$ be an instance to SUBSETSUM.

We denote $B^* := s_1 + \cdots + s_n$. Moreover, we may assume that $B \leq B^*$. Why?

Instance to PARTITION

- $n + 2$ numbers, namely
- $s_1, \ldots, s_n$,
- $B_1 := 2B^* - B \geq 1$,
- $B_2 := B^* + B$.

These numbers add up to $B^* + (2B^* - B) + (B^* + B) = 4B^*$.

Hence, a subset A' satisfying (2) has sum $2B^*$.

Correctness of the Reduction: "$\Rightarrow$"

Construction is doable in time being polynomial in input size.

Assume instance to SubsetSum is YES-instance. Then:

- There is a subset $A \subseteq \{1, \ldots, n\}$ with sum B.
- Define A' by adding $n+1$, the index of the special number $B_1 = 2B^* - B$, to A.
- Elements from A' add up to $2B^*$.
- Hence, instance to Partition is YES-instance.

Correctness of the Reduction: "$\Leftarrow$"

Assume there is a subset A satisfying (2) for the constructed PARTITION instance.

Exercise: *Either* B_1 *or* B_2 must belong to A.

If B_1 belongs to A, then consider A. Otherwise, we know that B_2 is in A and consider the numbers not chosen by A (including B_1) instead. They are also a solution with sum $2B^*$.

Hence, w.l.o.g., $B_1 := 2B^* - B$ is chosen by the solution. The remaining numbers (a subset of $\{s_1, \ldots, s_n\}$) sum up to B.

Hence, the remaining numbers represent a solution for the original SUBSETSUM instance.

Altogether: Instance to SUBSETSUM is a YES-instance if and only if instance to PARTITION is a YES-instance. Reduction PROVED.

On the Project

Today: project assignment handed out.

- Submission deadline: 2nd November (3 weeks + fall break)
- Counts as three ordinary exercise sheets (max. 6 points).
- Contains a theoretical and a practical part.
- Should be worked on in groups of size 2–3 (3 hard maximum).
- Please form groups (e. g. at the exercise session today) and register them on DTU Learn. Deadline: tomorrow (Wednesday) at 23.59.
- Note: you cannot hand in your solution if you are not registered in a group on DTU Learn.

VertexCover

Problem [VertexCover]
Input: An undirected graph $G = (V, E)$ and a natural number k.
Output: YES if there is a set $V' \subseteq V$ such that $|V'| \leq k$ and V' covers the edges, i.e., for all $e = \{v, w\} \in E$: $(v \in V') \vee (w \in V')$. NO, otherwise.

Theorem

The problem VertexCover *is $\mathcal{NP}$-complete.*

Sketch of Proof

To prove that VertexCover (VC) is in $\mathcal{NP}$ we interpret the the random R string as a specification of the k vertices of a cover and check whether this is the case.

To prove $\mathcal{NP}$-completeness, we reduce from 3-Sat: $\text{3-Sat} \leq_p \text{VC}$

The proof is by component design:

- Given an instance (X, C) of 3-Sat,
- Construct an instance $(G = (V, E), k)$ of VC,
- G is built of many small components,
- some ensure a consistent assignment, some satisfiability of the clauses.

Constructing G

Let $X = \{x_1, \ldots, x_n\}$ be the Boolean variables and
$C = \{c_1, \ldots, c_m\}$ the 3-clauses over X.

We set $k := n + 2m$ for the Vertex Cover instance.

The vertex set of G is:

$$V = \{x_1, \ldots, x_n\} \cup \{\bar{x}_1, \ldots, \bar{x}_n\} \cup \bigcup_{j=1}^{m} \{a_1(j), a_2(j), a_3(j)\}$$

Consider the following example with $n = 4$ and $m = 2$:
$X = \{x_1, x_2, x_3, x_4\}$, $C = \{c_1, c_2\}$,
$c_1 = x_1 \vee \bar{x}_3 \vee \bar{x}_4$, $c_2 = \bar{x}_1 \vee x_2 \vee \bar{x}_4$.

Then $V =$
$\{x_1, x_2, x_3, x_4, \overline{x}_1, \overline{x}_2, \overline{x}_3, \overline{x}_4, a_1(1), a_2(1), a_3(1), a_1(2), a_2(2), a_3(2)\}$

Constructing G

The *truth setting components* T_i are the edges

$$\forall x_i \in X \quad \text{let } T_i \text{ be the edge } T_i = \{x_i, \bar{x}_i\}$$

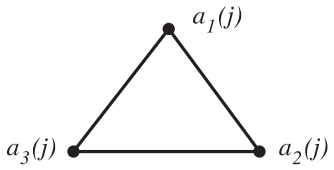
Idea: in a minimal Vertex Cover, only one vertex is chosen. This determines the truth setting: x_i chosen means $x_i = \text{true}$.

This gives n vertices to be chosen for the cover.

Constructing G

The *components for clause satisfiability* are defined as follows.

$$\forall c_j \in C\text{: let } S_j \text{ be the set of edges}$$
$$S_j = \{\{a_1(j), a_2(j)\}, \{a_2(j), a_3(j)\}, \{a_3(j), a_1(j)\}\}$$



Idea: At least two vertices per triangle are needed to cover it. The one not chosen corresponds to the "true" literal which satisfies c_j.

This makes another $2m$ vertices to be chosen for the cover.

Constructing G

The *connecting components* are defined as follows:

Let $c_j = z_1 \vee z_2 \vee z_3$,

z_s is a literal x_i or $\bar{x}_i$.

Add three edges:

$$K_j = \{\{a_1(j), z_1\},\ \{a_2(j), z_2\},\ \{a_3(j), z_3\}\}\,.$$

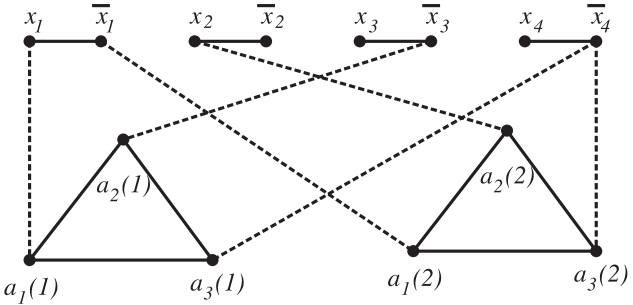
Idea: Connect the occurrences of the variables in the clauses to the truth setting. Result: all occurrences get the same truth value.

Finally: Construction is doable in time polynomial in n and m.

Constructing G: Example

$X = \{ x_1, x_2, x_3, x_4 \}$

$C = \{ c_1, c_2 \}$ $\quad c_1 = x_1 \vee \overline{x}_3 \vee \overline{x}_4 \quad c_2 = \overline{x}_1 \vee x_2 \vee \overline{x}_4$



$k = 4 + 2 \cdot 2 = 8$

Correctness of the Reduction

Show that $G = (V, E)$ has a Vertex Cover of size at most k iff the clauses of C have a satisfying assignment.

$\Rightarrow$

Let us first assume that $\alpha\colon X \to \{0,1\}$ is an assignment satisfying all clauses c_j.

We construct Vertex Cover $V' \subseteq V$ in two steps.

First select n vertices:

$$x_i \in V' \iff \alpha(x_i) = 1$$
$$\bar{x}_i \in V' \iff \alpha(x_i) = 0$$

This way, all T_i-edges are covered.

Moreover, for every triangle S_j, at least one connecting edge is covered, say $\{a_1(j), z_k\}$.

$\Rightarrow$

Put $a_2(j)$ and $a_3(j)$ into V'.

This covers S_j as well as the two other connecting edges.

The set V' covers all edges and has size $k = n + 2m$.

"$\Rightarrow$" direction has been shown.

Still showing that $G = (V, E)$ has a Vertex Cover of size at most k iff the clauses of C have a satisfying assignment.

Let us now assume that $V' \subseteq V$ is a Vertex Cover for G of size $|V'| \leq k = n + 2m$.

In order to cover the edges $T_i = \{x_i, \bar{x}_i\}$, at least one of x_i or $\bar{x}_i$ has to be in V'. Needs n vertices ($2m$ left).

In order to cover each of the m triangles, at least two vertices of each one have to be in V'.

All vertices of V' are used. Since these are only $n + 2m$ many:

- from every triangle two vertices are in V',
- from every truth setting component exactly one is in V'.

Also every connecting edge K_j is covered by V'.

Define the assignment $\alpha\colon X \to \{0, 1\}$

$$\alpha(x_i) = \begin{cases} 1 \text{ if } x_i \in V' \\ 0 \text{ if } \bar{x}_i \in V' \end{cases}$$

Note: As V' contains exactly one of x_i or $\bar{x}_i$,
the assignment is well defined.

We next show that all clauses are satisfied by this assignment.

- Let $c_j = z_1 \vee z_2 \vee z_3$ be some clause.
- Exactly two vertices of S_j belong to V', say $a_1(j)$, $a_2(j)$.
- These cover the connecting edges $\{a_1(j), z_1\}$ and $\{a_2(j), z_2\}$.
- The third connecting edge $\{a_3(j), z_3\}$ is then covered by vertex z_3 of T_i.
- Assignment α assigns true to z_3.
- By construction of G, the vertex z_3, viewed as a literal, is the third literal in c_j.
- Clause c_j is satisfied.
- As c_j was arbitrarily selected, all clauses are satisfied. □

Problems Related with VertexCover

Problem [Clique]
Input: An undirected graph $G = (V, E)$ and a natural number k.
Output: YES if there is a set $V' \subseteq V$ such that $|V'| \geq k$ and V' is a clique, i.e., for all $v, w \in V'$, where $v \neq w$, it holds $\{v, w\} \in E$. NO otherwise.

Problem [IndependentSet]
Input: An undirected graph $G = (V, E)$ and a natural number k.
Output: YES if there is a set $V' \subseteq V$ such that $|V'| \geq k$ and V' is an independent set, i.e., for all $v, w \in V'$, where $v \neq w$, it holds $\{v, w\} \notin E$. NO otherwise.

Reductions VertexCover $\leq_p$ Clique $\leq_p$ IndependentSet are not too difficult (see lecture notes).