MANCHESTER 1824
The University of Manchester

COMP26120: Tractability and NP Completeness (2018/19)

Lucas Cordeiro

lucas.cordeiro@manchester.ac.uk

Reduction

- The crux of NP-Completeness is **reducibility**

Reduction

- The crux of NP-Completeness is **reducibility**
  - A problem ***A*** can be reduced to another problem ***B*** if any instance ***α*** of ***A*** can be transformed into some instance of ***β*** of ***B***:
    - The **transformation** takes **polynomial-time**
    - The answer for ***α*** is *"yes" iff* the answer for ***β*** is also *"yes"*

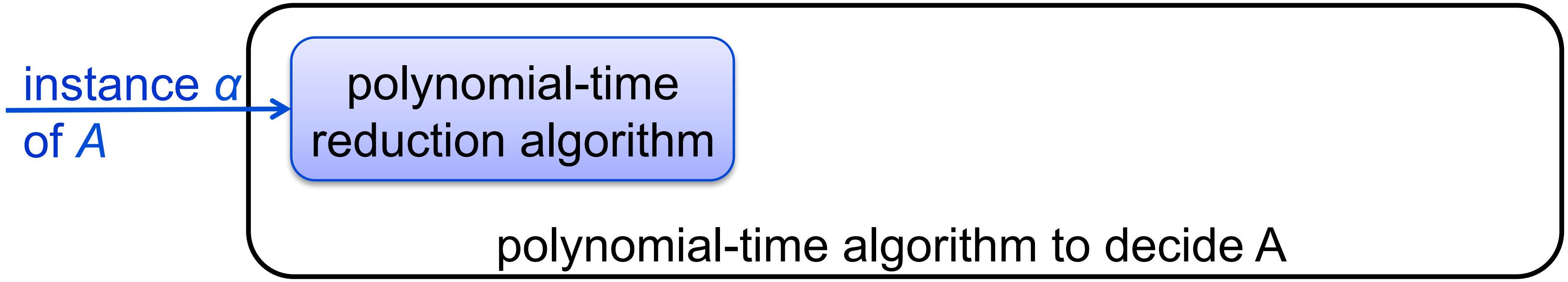
Reduction

- The crux of NP-Completeness is **reducibility**
  - A problem ***A*** can be reduced to another problem ***B*** if any instance ***α*** of ***A*** can be transformed into some instance of ***β*** of ***B***:
    - The **transformation** takes **polynomial-time**
    - The answer for ***α*** is *"yes" iff* the answer for ***β*** is also *"yes"*
  - If ***A*** reduces to ***B***, ***A*** is "no harder to solve" than ***B***

Reduction

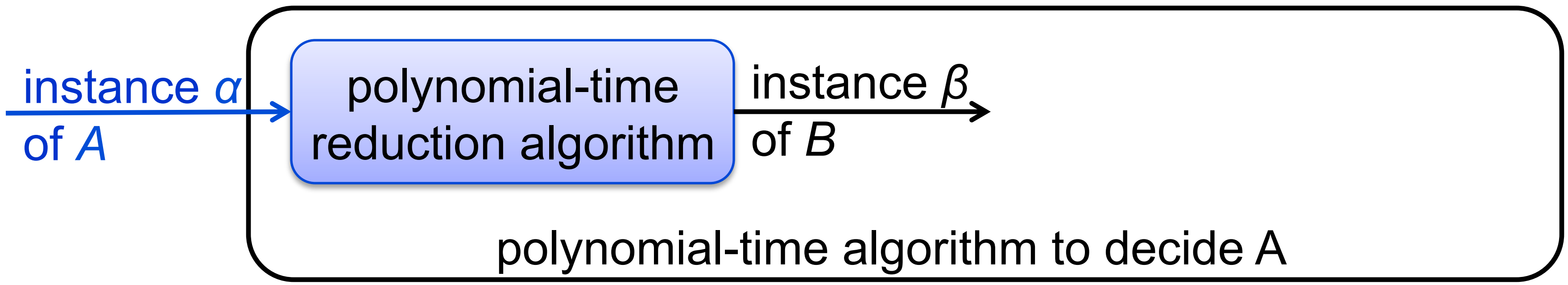
- The crux of NP-Completeness is **reducibility**
  - A problem ***A*** can be reduced to another problem ***B*** if any instance ***α*** of ***A*** can be transformed into some instance of ***β*** of ***B***:
    - The **transformation** takes **polynomial-time**
    - The answer for ***α*** is *"yes" iff* the answer for ***β*** is also *"yes"*
  - If ***A*** reduces to ***B***, ***A*** is "no harder to solve" than ***B***
  - We are trying to prove that **no efficient algorithm is likely to exist**

Polynomial-time Reduction



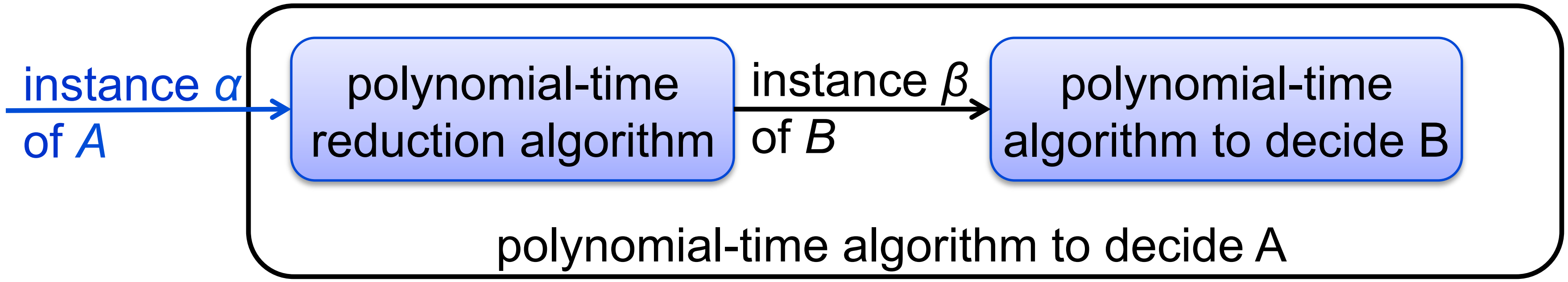
1) Given an instance α of problem A, use a polynomial-time reduction algorithm

Polynomial-time Reduction



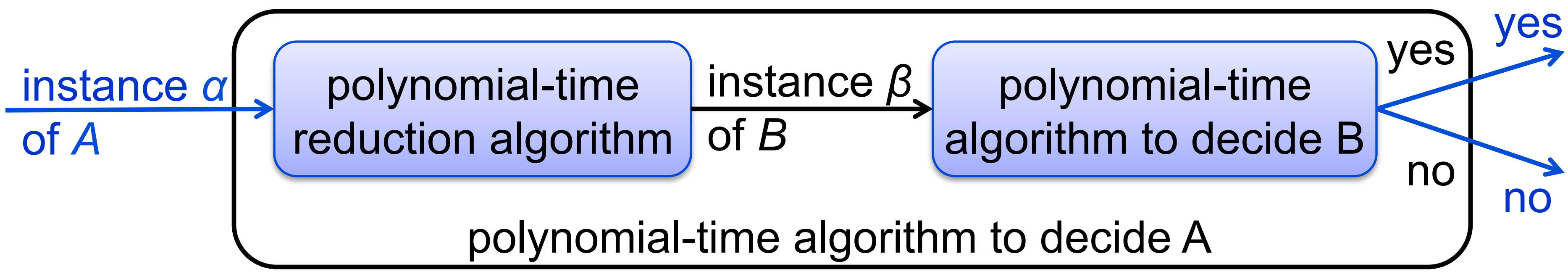
1) Given an instance α of problem A, use a polynomial-time reduction algorithm
2) Transform it to an instance β of problem B

Polynomial-time Reduction



1) Given an instance α of problem A, use a polynomial-time reduction algorithm
2) Transform it to an instance β of problem B
3) Run the polynomial-time decision algorithm for B on the instance β

Polynomial-time Reduction



1) Given an instance α of problem *A*, use a polynomial-time reduction algorithm
2) Transform it to an instance β of problem *B*
3) Run the polynomial-time decision algorithm for *B* on the instance β
4) Use the answer for β as the answer for α

Proving NP-Completeness

- Prove NP-completeness of a language *L* by reduction consists of the following steps
  1. Prove **L** $\in$ ***NP***

Proving NP-Completeness

- Prove NP-completeness of a language *L* by reduction consists of the following steps
  1. Prove **L** $\in$ ***NP***
  2. Select a known **NP-complete language L'**

Proving NP-Completeness

- Prove NP-completeness of a language *L* by reduction consists of the following steps
  1. Prove **L** $\in$ ***NP***
  2. Select a known **NP-complete language L'**
  3. Describe an algorithm that computes a **function *f*** mapping every instance x $\in$ *{0, 1}*of L' to an instance f(x) of L*

Proving NP-Completeness

- Prove NP-completeness of a language *L* by reduction consists of the following steps
  1. Prove **L** $\in$ ***NP***
  2. Select a known **NP-complete language L'**
  3. Describe an algorithm that computes a **function *f*** mapping every instance x $\in$ *{0, 1}*of L' to an instance f(x) of L*
  4. Prove that the function *f* satisfies $x \in L'$ *iff* $f(x) \in L$ for all $x \in \{0,1\}^*$

Proving NP-Completeness

- Prove NP-completeness of a language *L* by reduction consists of the following steps
  1. Prove **L** $\in$ ***NP***
  2. Select a known **NP-complete language L'**
  3. Describe an algorithm that computes a **function *f*** mapping every instance x $\in$ *{0, 1}*of L' to an instance f(x) of L*
  4. Prove that the function *f* satisfies *x* $\in$ *L' iff f(x)* $\in$ *L* for all *x* $\in$ *{0,1}**
  5. Prove that the algorithm computing ***f*** runs in **polynomial-time**

1.5 hours

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

COMP26120

January 2018

Time: 00:00 – 00:00

Marking Scheme Included

Do not publish

Answer both questions

The use of electronic calculators is permitted provided they are not programmable and do not store text.

---

1. Tractability and NP Completeness

a) **(NP Completeness)** A clique in an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices, each pair of which is connected by an edge in E, i.e., a clique is a complete subgraph of G. The size of a clique is the number of vertices it contains. The problem of finding a clique of maximum size in a graph is NP-complete. The decision problem CLIQUE has the corresponding language:

$$CLIQUE = \{\langle G, k\rangle : G \text{ is a graph containing a clique of size } k\}$$

Sketch a proof of NP-completeness for the decision problem CLIQUE.

No marks will be given for simply stating that CLIQUE is NP-complete.

(10 marks)

Model Answer:
1. **Prove CLIQUE $\in$ NP**.
For a given $G = (V, E)$, we use the set $V' \subseteq V$ of vertices in the clique as a certificate for G. We can check whether V' is a clique in polynomial time by checking whether for each pair $u, v \in V'$, the edge (u, v) belongs to E.
2. **Select Satisfiability of boolean formulas in 3-CNF as a known NP-complete language called 3-CNF-SAT**.
3. **Describe an algorithm that computes a function f mapping every instance of 3-CNF-SAT to an instance of CLIQUE**.
Let $\phi = C_1 \wedge C_2 \wedge \ldots C_k$ be a boolean formula in 3-CNF with k clauses. For $r = 1, 2, \ldots, k$, each clause C_r has exactly three distinct literals l_1^r, l_2^r, and l_3^r. For each clause $C_r = (l_1^r \vee l_2^r \vee l_3^r)$ in ϕ, we place a triple of vertices v_1^r, v_2^r, and v_3^r into V. We put an edge between two vertices v_i^r and v_j^s if both of the following hold:

- v_i^r and v_j^s are in different triples, i.e., $r \neq s$, and
- their corresponding literals are consistent, i.e., l_i^r is not the negation of l_j^s.

4. **Show that this transformation of ϕ into G is a reduction**.
First, supposed that ϕ has a satisfying assignment. Then each clause C_r contains at least one literal l_i^r that is assigned 1, and each such literal corresponds to a vertex v_i^r. Picking one such "true" literal from each clause yields a set V' of k vertices. We claim that V' is a clique. For any two vertices $v_i^r, v_j^s \in V'$, where $r \neq s$ both corresponding literals l_i^r and l_j^s map to 1 by the given satisfying assignment, and thus the literals cannot be complements. Thus, by the construction of G, the edge (v_i^r, v_j^s) belongs to E.
5. **Show that the algorithm computing f runs in polynomial time**.
We can easily build this graph from ϕ in polynomial time. As an example of this construction, if we have
$\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$ then G is the graph shown below.
Distribution of Marks: 2 marks for each step of the proof. No marks for answer without explanation.

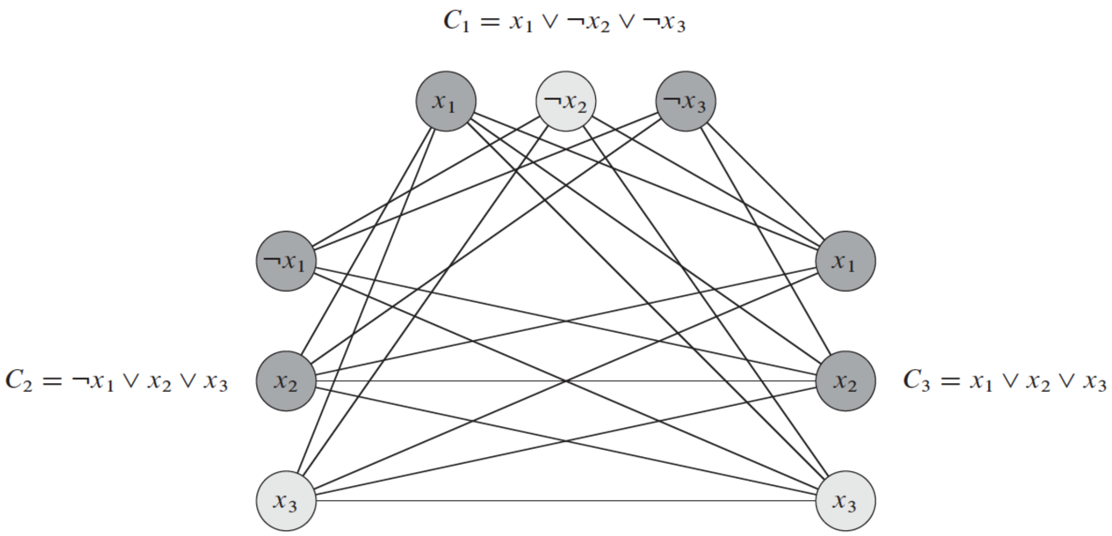


Figure 1: The graph G derived from the 3-CNF formula ϕ. A SAT assignment has $x_2 = 0$, $x_3 = 1$, and x_1 either 0 or 1. This assignment satisfies C1 with $\neg x_2$, and it satisfies C_2 and C_3 with x_3, corresponding to the clique with lightly shaded vertices.

END OF EXAMINATION