Formale Methoden der Informatik

Block 1: Foundations of Complexity Theory

4. NP-Completeness

Reinhard Pichler

Institut für Informationssysteme
Arbeitsbereich DBAI
Technische Universität Wien

16 October, 2013

Outline

Variants of Satisfiability

SAT

INSTANCE: Boolean formula φ.

QUESTION: Is φ satisfiable?

3-SAT

INSTANCE: Boolean formula φ in 3-CNF

QUESTION: Is φ satisfiable?

2-SAT

INSTANCE: Boolean formula φ in 2-CNF

QUESTION: Is φ satisfiable?

Variants of Satisfiability

- We have already seen that **2-SAT** is tractable:
  A polynomial-time algorithm can, for instance, be obtained via a polynomial-time Turing reduction to **REACHABILITY**.
- In contrast, **SAT** and **3-SAT** are intractable:
  In this lecture, we shall only briefly mention the proof idea of the NP-completeness proof for these two problems. Detailed proofs will be given in the Komplexitätstheorie lecture in the summer term.

Complexity of SAT and 3-SAT

Cook-Levin Theorem

SAT is NP-complete.

Theorem

3-SAT is NP-complete.

Proof of the NP-membership

SAT and also **3-SAT** can be decided by the following NP-algorithm:

1. Guess a truth assignment T for the variables in φ.
2. Check that φ is **true** in T.

Proof idea of the NP-hardness of **SAT**

To show that **SAT** is NP-hard, we have to show that for every problem $\mathcal{P}$ in NP there is a polynomial-time many-one reduction from $\mathcal{P}$ to **SAT**.

Since NP has infinitely many problems, we cannot deal with each problem separately. We need a method to deal with all problems in NP at once.

The idea is as follows. Take an arbitrary problem $\mathcal{P} \in$ NP and an arbitrary instance I of $\mathcal{P}$. We must show that we can build in polynomial time a formula φ such that I is a positive instance of $\mathcal{P} \Leftrightarrow \varphi$ is satisfiable.

Since $\mathcal{P} \in$ NP, by definition there must exist a polynomially balanced and polynomially decidable certificate relation R for $\mathcal{P}$. Recall that I is a positive instance of $\mathcal{P} \Leftrightarrow$ there exists an object C of polynomial size in $|I|$ such that $(I, C) \in R$. For testing $(I, C) \in R$ we have a polynomial time algorithm A available.

Proof idea of the NP-hardness of **SAT**

Thus to provide the reduction it suffices to build in polynomial time a formula φ such that the following equivalence holds: φ is satisfiable $\Leftrightarrow$ there is some polynomial-size object C such that A returns *true* on (I, C).

Intuitively, the desired propositional formula φ must perform two jobs:

1. generate the candidate certificates,
2. simulate the computation of A on a candidate pair (I, C).

The second part is non-trivial: how can we use a Boolean formula to simulate SIMPLE programs? How to deal with **while** loops, **for** loops, **if**/**then**/**else** statements?

Answer: this can be done in principle, but it makes more sense to substitute SIMPLE programs by Turing machines. Turing machines are a much simpler model of computation that is still equivalent to SIMPLE in terms of computability and complexity.

Complexity of SAT and 3-SAT

Proof idea of the NP-hardness of **3-SAT**

We have to reduce **SAT** to **3-SAT**, i.e.: Let φ be an arbitrary Boolean formula. We have to show that there exists a Boolean formula $R(\varphi) = \psi$, s.t. ψ is in 3-CNF and φ is satisfiable $\Leftrightarrow$ ψ is satisfiable.

Remarks.

- An arbitrary Boolean formula φ can be transformed in polynomial time into a sat-equivalent formula ψ in 3-CNF.
- In general, φ and ψ are not logically equivalent.
- This result is by no means trivial: The "usual" transformation into CNF via de Morgan's laws and the distributivity of $\wedge$ and $\vee$ usually leads to an exponential blow-up. For instance, consider the CNF which is logically equivalent to $(x_1 \wedge y_1) \vee \ldots \vee (x_n \wedge y_n)$.

The VALIDITY Problem

VALIDITY

INSTANCE: Boolean formula φ.

QUESTION: Is φ valid (i.e., **true** in every assignment appropriate to φ)?

Definition

The class co-NP is the class of problems whose complement is in NP, i.e., a problem $\mathcal{P}$ is in co-NP iff the complement co-$\mathcal{P}$ of $\mathcal{P}$ is in NP.

Theorem

VALIDITY *is* co-NP-*complete (i.e., co-***VALIDITY** *is* NP-*complete).*

Proof

Recall the following equivalences:
φ is valid $\Leftrightarrow$ $\neg\varphi$ is unsatisfiable and φ is unsatisfiable $\Leftrightarrow$ $\neg\varphi$ is valid.

Membership. **VALIDITY** can be reduced to the co-**SAT**-problem.
Since **SAT** is in NP, co-**SAT** is in co-NP and so is **VALIDITY**.

Hardness. co-**SAT** can be reduced to the **VALIDITY**-problem.
Since **SAT** is NP-hard, co-**SAT** is co-NP-hard and so is **VALIDITY**.

Some NP-complete Graph Problems

We have already encountered the **INDEPENDENT SET** problem. The following two problems are closely related:

CLIQUE

INSTANCE: Undirected graph $G = (V, E)$ and integer K.

QUESTION: Does there exist a *clique* C of size $\geq K$?
i.e., $C \subseteq V$, s.t. for all $i, j \in C$ with $i \neq j$, $[i, j] \in E$.

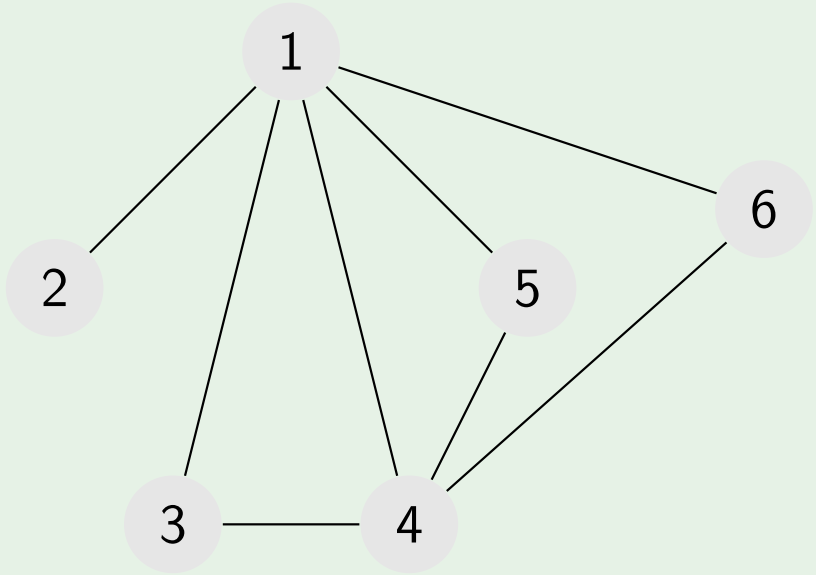
VERTEX COVER

INSTANCE: Undirected graph $G = (V, E)$ and integer K.

QUESTION: Does there exist a *vertex cover* N of size $\leq K$?
i.e., $N \subseteq V$, s.t. for all $[i, j] \in E$, either $i \in N$ or $j \in N$.

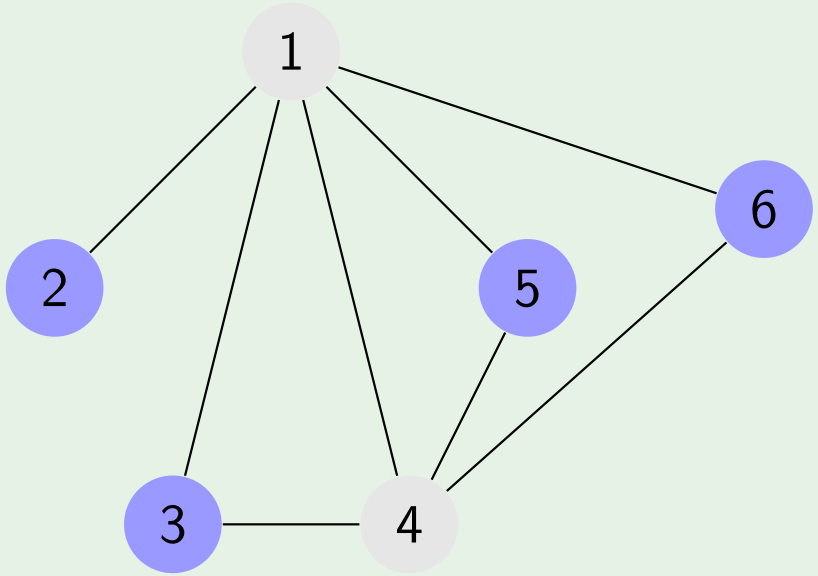
INDEPENDENT SET vs. CLIQUE

Example



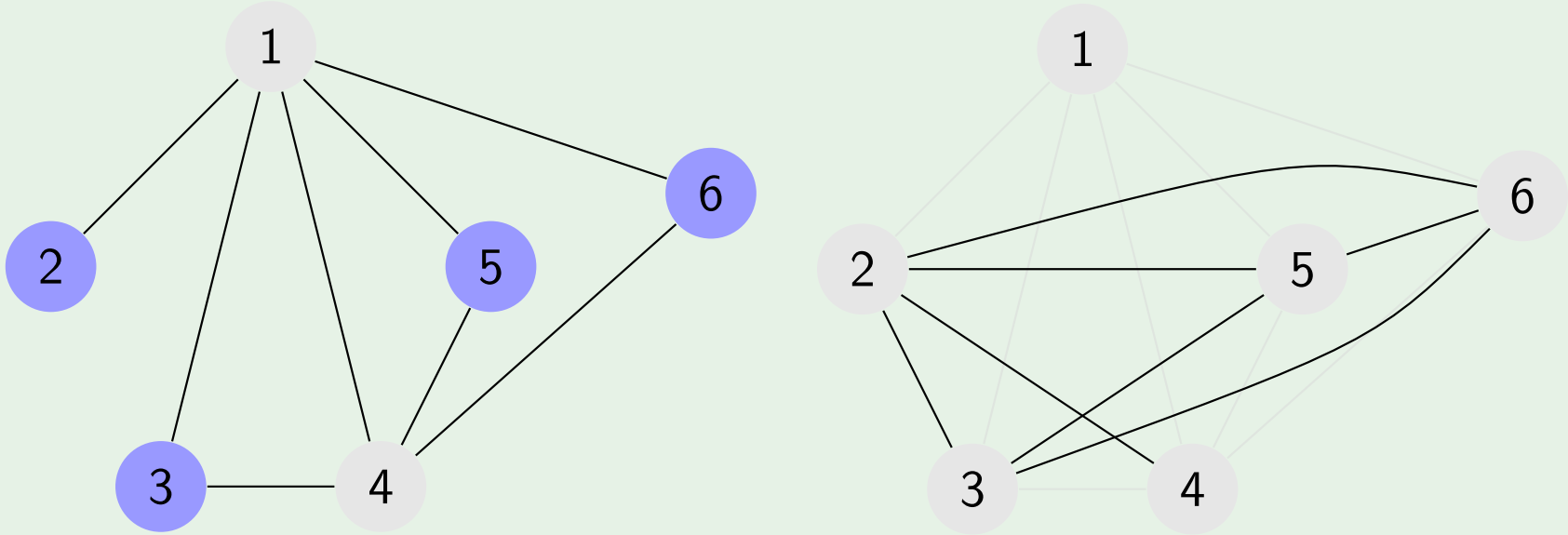
INDEPENDENT SET vs. CLIQUE

Example



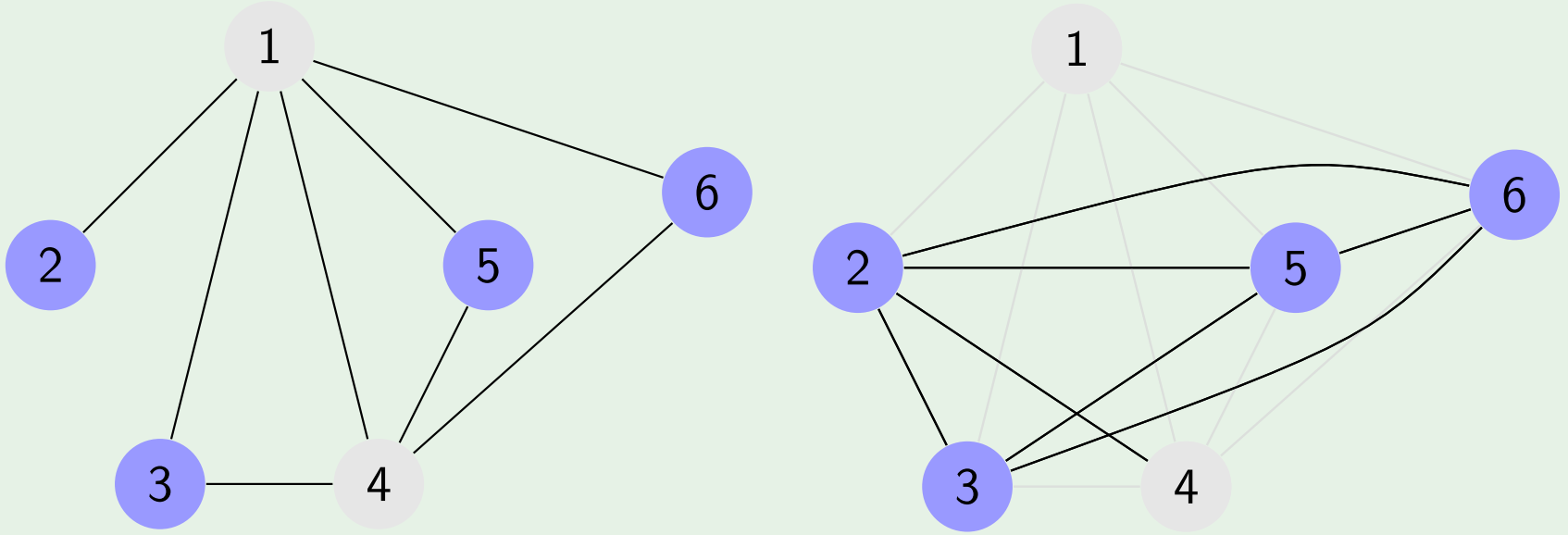
INDEPENDENT SET vs. CLIQUE

Example



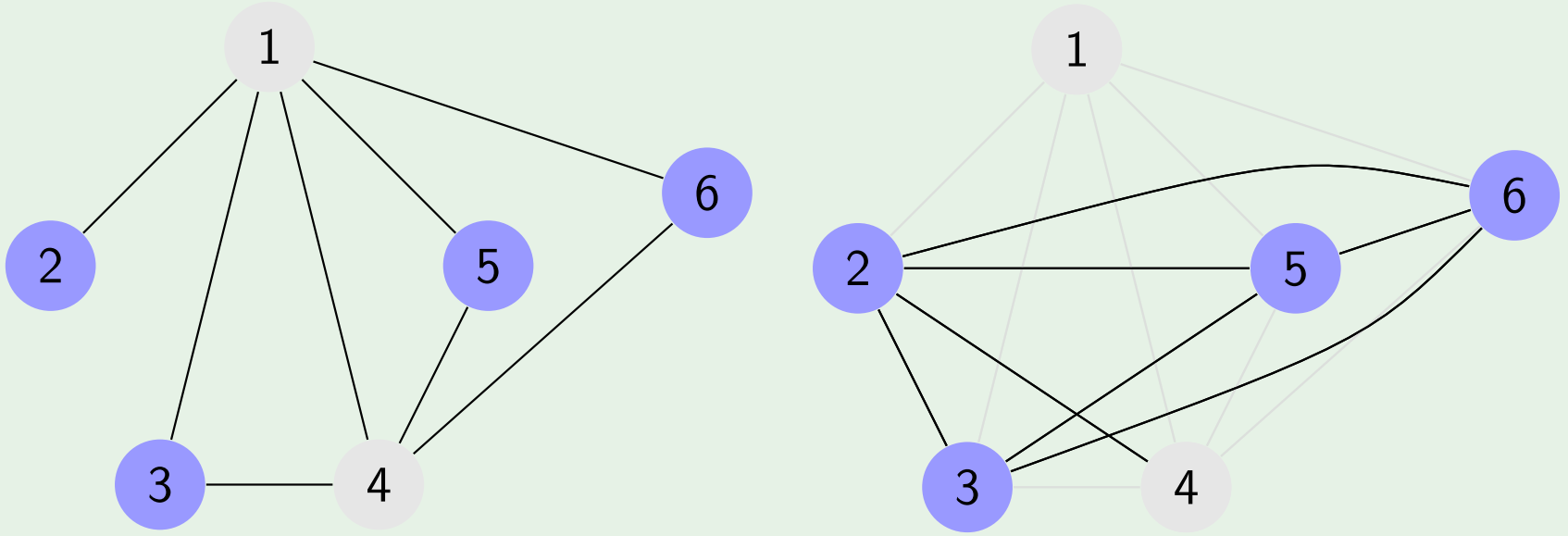
INDEPENDENT SET vs. CLIQUE

Example



INDEPENDENT SET vs. CLIQUE

Example

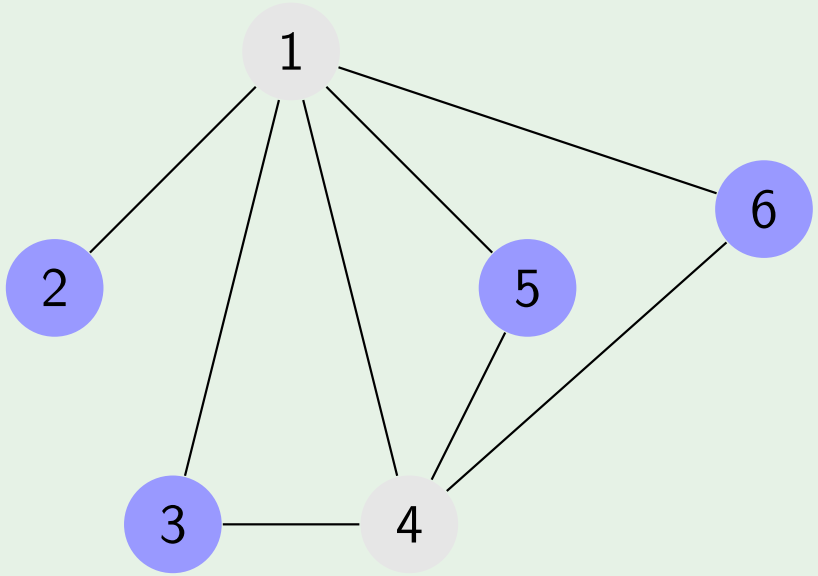


Proposition

Let $G = (V, E)$ be an undirected graph with $I \subseteq V$. Moreover, let $\overline{G} = (V, \overline{E})$ be the complement graph, i.e. $[i, j] \in E \Leftrightarrow [i, j] \notin \overline{E}$.
I is an independent set in $G \Leftrightarrow I$ is a clique in $\overline{G}$.

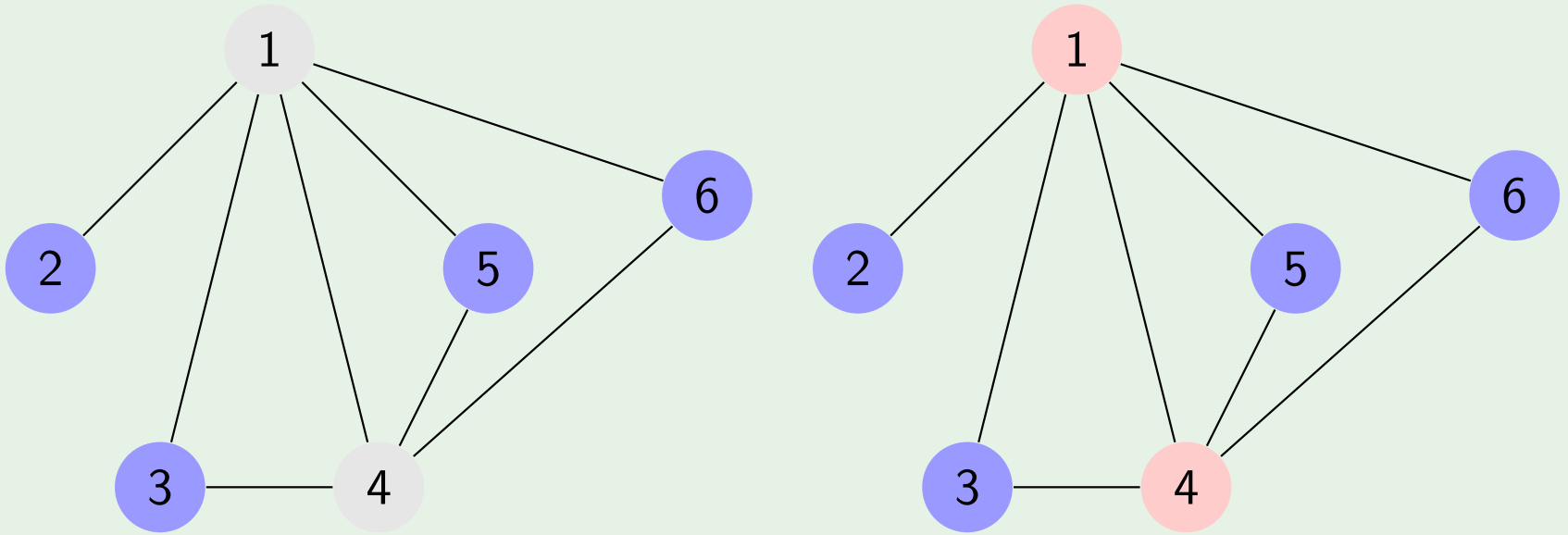
INDEPENDENT SET vs. VERTEX COVER

Example



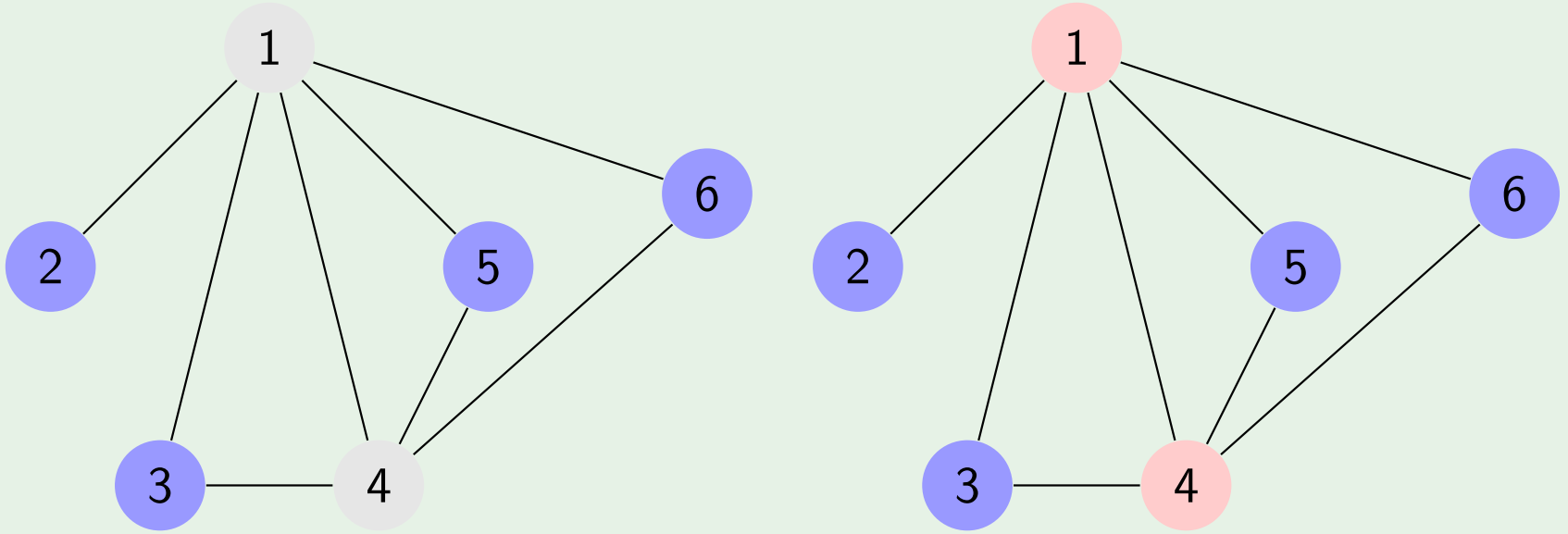
INDEPENDENT SET vs. VERTEX COVER

Example



INDEPENDENT SET vs. VERTEX COVER

Example



Proposition

Let $G = (V, E)$ be an undirected graph with $I \subseteq V$.
I is an independent set in $G \Leftrightarrow N = V \setminus I$ is a vertex cover in G.

Idea. An independent set never contains both endpoints of an edge.
Hence, of every edge in E, at least one endpoint is in $V \setminus I$.

Complexity

Theorem

INDEPENDENT SET, **CLIQUE**, *and* **VERTEX COVER** *are* NP-*complete.*

Proof

Membership. An NP-algorithm for these problems first guesses a subset S of the vertices V and then checks in polynomial time that S has the desired property (e.g., S is an independent set of size $\geq K$).

Hardness. By the above equivalences, it suffices to prove the NP-hardness of one of these 3 problems. In fact, we have already seen a reduction from **3-SAT** to **INDEPENDENT SET**, from which its NP-hardness follows immediately.

Further Graph Problems

3-COLORABILITY

INSTANCE: Undirected graph $G = (V, E)$

QUESTION: Does G have a *3-coloring*? i.e., an assignment of one of 3 colors to each of the vertices in V such that any two vertices i, j connected by an edge $[i, j] \in E$ do not have the same color?

k-COLORABILITY (for fixed value $k \geq 1$)

INSTANCE: Undirected graph $G = (V, E)$

QUESTION: Does G have a *k-coloring*? i.e., an assignment of one of k colors to each of the vertices in V such that any two vertices i, j connected by an edge $[i, j] \in E$ do not have the same color?

Further Graph Problems

HAMILTON-PATH

INSTANCE: (directed or undirected) graph $G = (V, E)$

QUESTION: Does G have a *Hamilton path*?
i.e., a path visiting all vertices of G exactly once.

HAMILTON-CYCLE

INSTANCE: (directed or undirected) graph $G = (V, E)$

QUESTION: Does G have a *Hamilton cycle*?
i.e., a cycle visiting all vertices of G exactly once.

Further Variants of Satisfiability

Not-all-equal SAT (NAESAT)

INSTANCE: Boolean formula φ in 3-CNF

QUESTION: Does there exist a truth assignment T on φ, such that the 3 literals in each clause do not have the same truth value?

1-IN-3-SAT

INSTANCE: Boolean formula φ in 3-CNF

QUESTION: Does there exist a truth assignment T on φ, such that in each clause, exactly one literal is **true** in T?

Remarks

- Clearly **1-IN-3-SAT** $\subset$ **NAESAT** $\subset$ **3-SAT**. The instances of these 3 problems are the same, namely 3-CNF formulae. However, the positive instances of **1-IN-3-SAT** are a proper subset of **NAESAT**, which in turn are a proper subset of the positive instances of **3-SAT**.
- Note that the NP-completeness of any of these 3 problems does not immediately imply the NP-completeness of any of the other problems, since it is a priori not clear if further constraining the positive instances makes things easier or harder.
- **3-SAT** is a special case of **SAT**, i.e., the instances of the former are a proper subset of the latter while the question remains the same. The NP-hardness of the special case immediately implies the NP-hardness of the general case.

NP-Completeness

Theorem

All of the following problems are NP*-complete.*

- **k-COLORABILITY** *for any* $k \geq 3$ *(e.g.,* **3-COLORABILITY***)*
- **HAMILTON-PATH**, **HAMILTON-CYCLE**, **TSP(D)**
- **k-SAT** *for any* $k \geq 3$, **NAESAT**, **1-IN-3-SAT**

Proof

Membership. All problems above can be solved by guessing a candidate of polynomial size and checking in polynomial time that it is a solution.

Hardness. in the Komplexitätstheorie lecture in the summer term

Problem Solving by Reductions to SAT

Motivation

- **SAT** is a canonical NP-complete problem, and thus is often used to prove NP-hardness of other problems by a reduction from **SAT**.
- However, reductions to **SAT** also play an important role because efficient **SAT** solvers have been developed in recent years.
- A reduction from a problem $\mathcal{P} \in$ NP to **SAT** together with a **SAT** solver may provide a good algorithm for $\mathcal{P}$.

Graph Homomorphism

HOMOMORPHISM

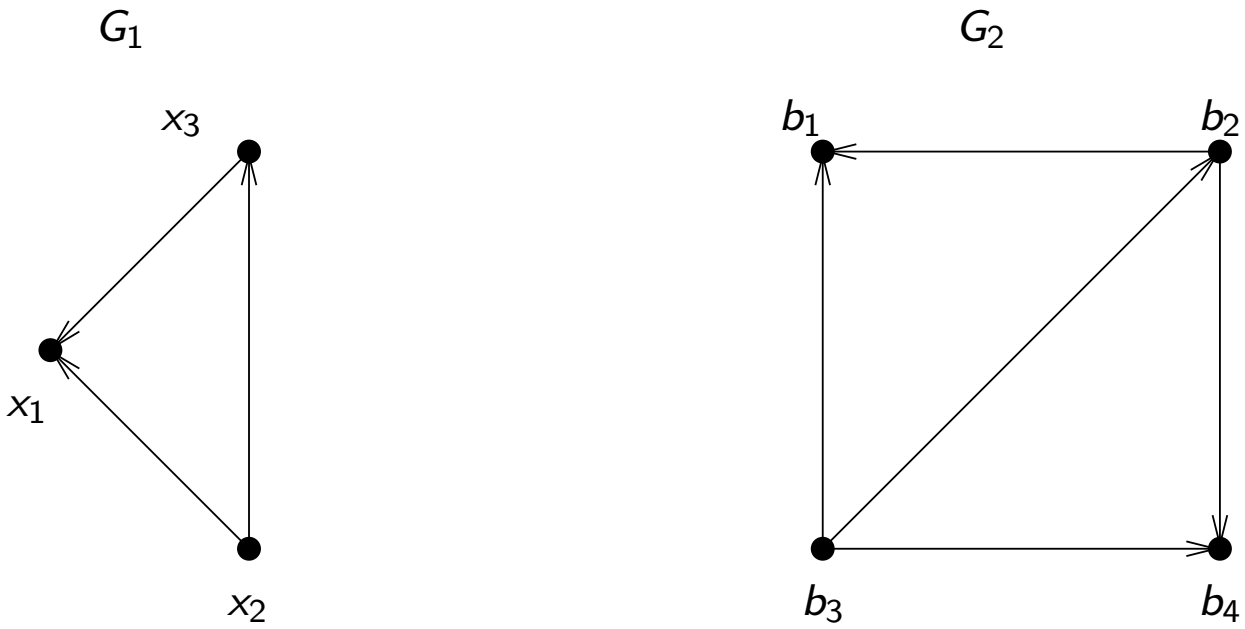
INSTANCE: Two directed graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$

QUESTION: Does there exist a homomorphism from G_1 to G_2? That is, does there exist a mapping $h : V_1 \to V_2$ such that: if $(v_1, v_2) \in E_1$, then $(h(v_1), h(v_2)) \in E_2$?

HOMOMORPHISM is one of the basic tasks in querying databases:

- G_1 can be seen as a query, and G_2 as a database.
- We ask if G_1 can be matched in G_2.
- Answering **select-from-where** queries of **SQL** is essentially **HOMOMORPHISM** in disguise.

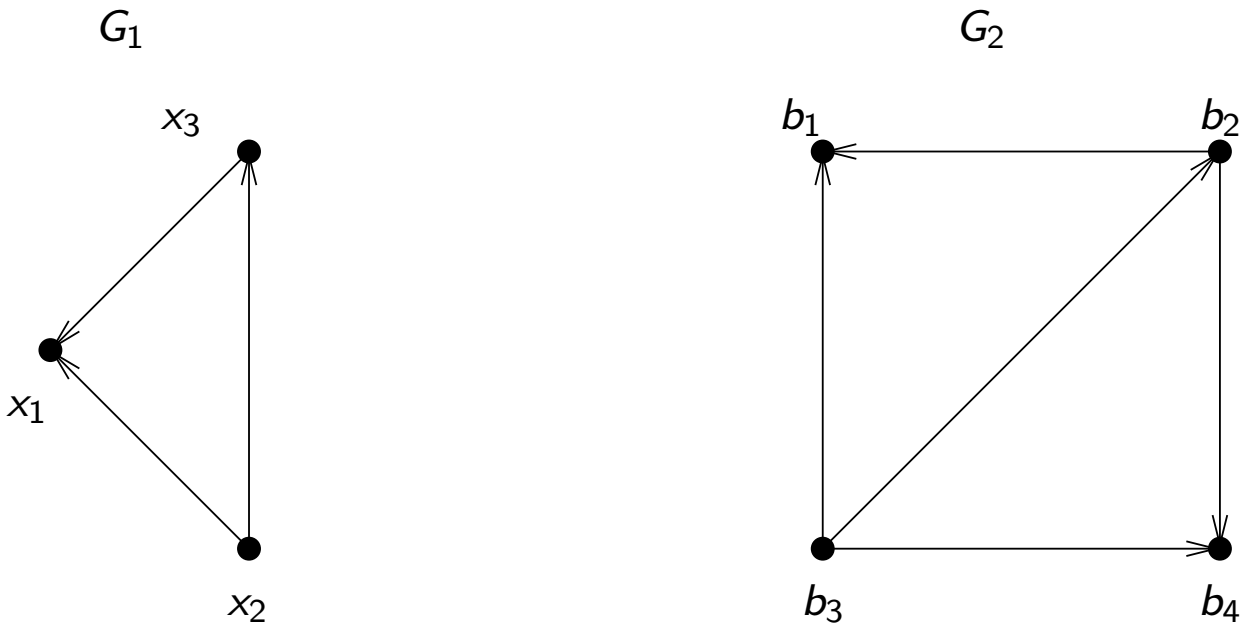
Graph Homomorphism: An Example (1)



G_2 can be seen as a database consisting of a single relation $E(\mathit{start}, \mathit{end})$:

$$E = \{(b_3, b_1), (b_2, b_1), (b_3, b_2), (b_2, b_4), (b_3, b_4)\}.$$

Graph Homomorphism: An Example (1)



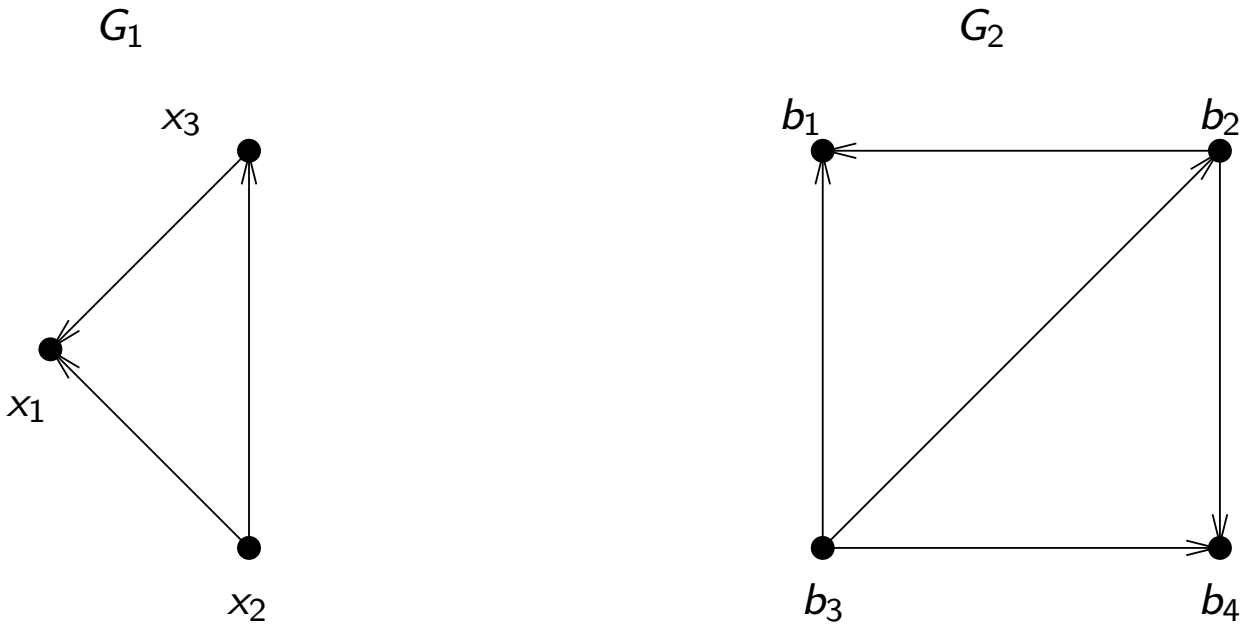
G_2 can be seen as a database consisting of a single relation $E(start, end)$:

$$E = \{(b_3, b_1), (b_2, b_1), (b_3, b_2), (b_2, b_4), (b_3, b_4)\}.$$

Computing all homomorphisms from G_1 to G_2 comes down to answering the following SQL query

```
select e1.end, e1.start, e2.start
from E e1, E e2, E e3
where e1.start = e3.start and e1.end = e2.end and e2.start = e3.end
```

Graph Homomorphism: An Example (1)



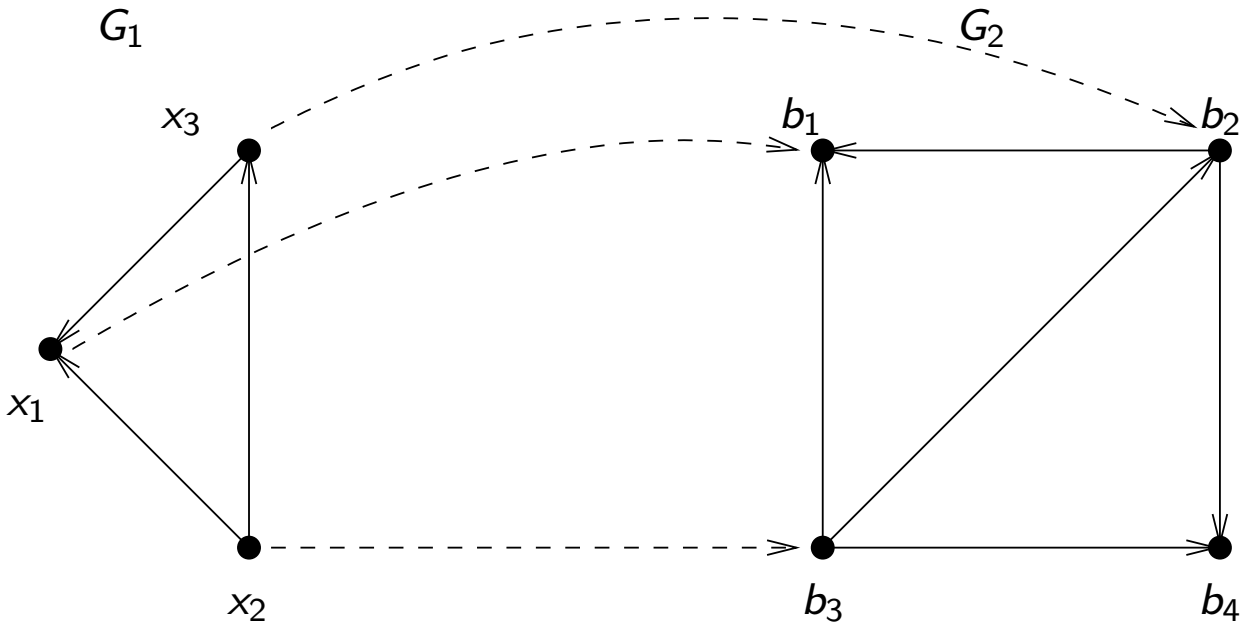
G_2 can be seen as a database consisting of a single relation $E(\mathit{start}, \mathit{end})$:

$$E = \{(b_3, b_1), (b_2, b_1), (b_3, b_2), (b_2, b_4), (b_3, b_4)\}.$$

More elegantly, we write this query in Datalog notation as

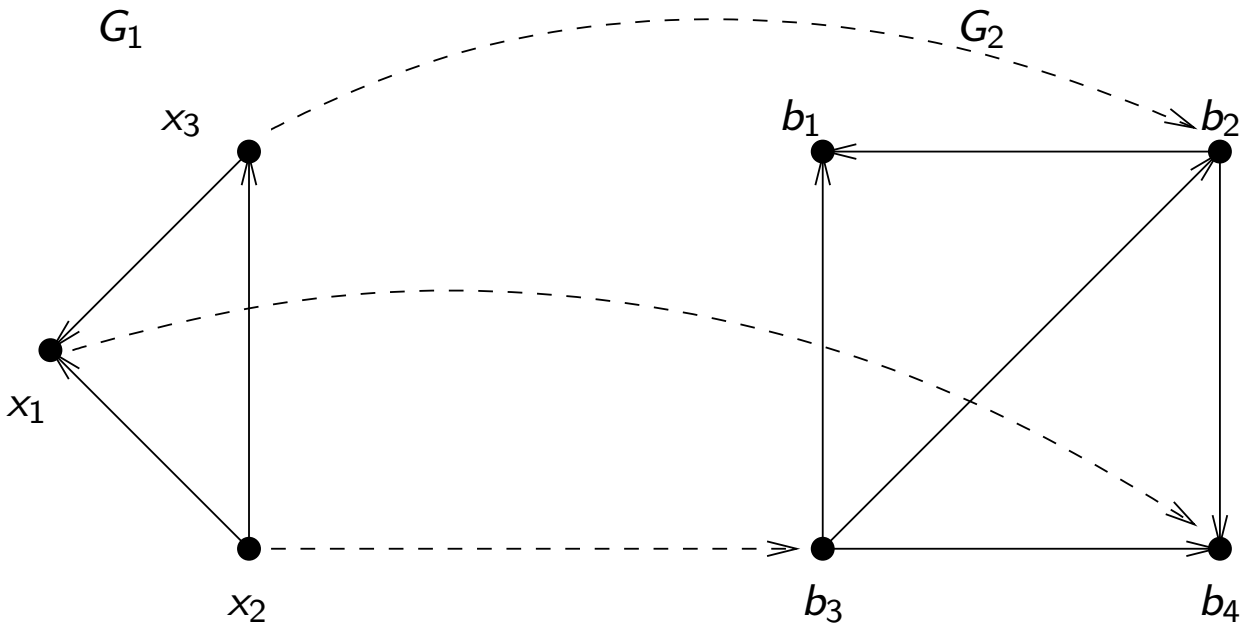
$$Q(x_1, x_2, x_3) :- E(x_2, x_1), E(x_3, x_1), E(x_2, x_3).$$

Graph Homomorphism: An Example (2)



The above homomorphism shows that (b_1, b_3, b_2) is in the answer of Q.

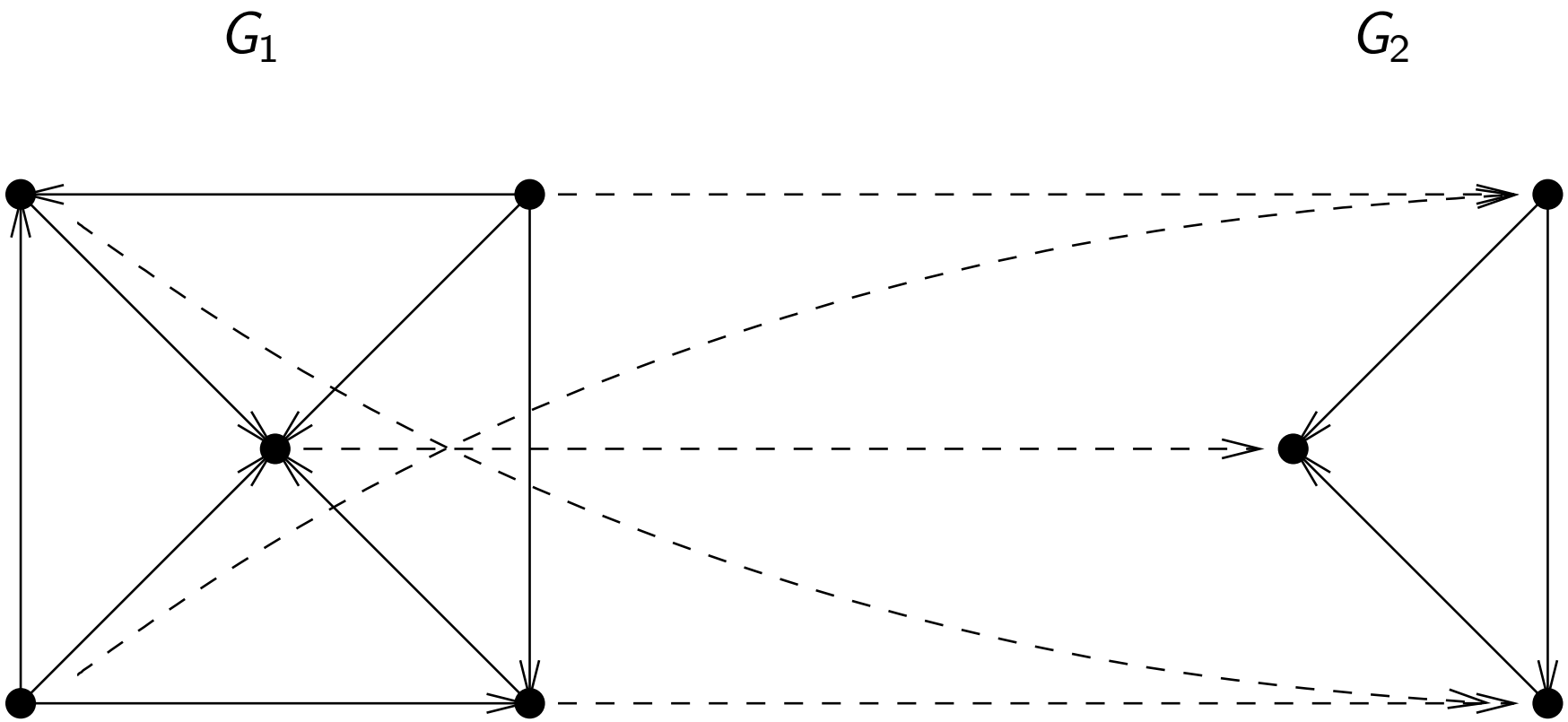
Graph Homomorphism: An Example (3)



The above homomorphism shows that (b_4, b_3, b_2) is in the answer of Q.

Are there any other homomorphisms (and answers to Q)? No.

Graph Homomorphism: A More Complicated Example



Reducing Homomorphism to SAT

Reduction

Let an arbitrary instance of HOMOMORPHISM be given by two directed graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Let $V_1 = \{a_1, \ldots, a_n\}$ and $V_2 = \{b_1, \ldots, b_m\}$.

We have to define a formula φ such that φ is satisfiable $\Leftrightarrow$ there exists a homomorphism from G_1 to G_2.

For φ we use the following propositional variables:

- M_{a_i,b_j} for each $1 \leq i \leq n$ and $1 \leq j \leq m$ (meaning: if M_{a_i,b_j} is *true*, then a_i of G_1 is mapped to b_j of G_2)
- $E^1_{a_i,a_j}$ for all $1 \leq i, j \leq n$ ($E^1_{a_i,a_j}$ means G_1 has an edge from a_i to a_j)
- $E^2_{b_i,b_j}$ for all $1 \leq i, j \leq m$ ($E^2_{b_i,b_j}$ means G_2 has an edge from b_i to b_j)

Reduction (continued)

The formula φ is defined as $\varphi = \alpha_1 \wedge \alpha_2 \wedge \alpha_3 \wedge \alpha_4 \wedge \alpha_5$, where

- α_1 says that each node in G_1 must be assigned to a node in G_2:

$$\alpha_1 = \bigwedge_{1 \leq i \leq n} \left(\bigvee_{1 \leq j \leq m} M_{a_i, b_j} \right)$$

- α_2 ensures that a node in G_1 is assigned to at most one node in G_2:

$$\alpha_2 = \bigwedge_{1 \leq i \leq n, 1 \leq k, l \leq m, k \neq l} (\neg M_{a_i, b_k} \vee \neg M_{a_i, b_l})$$

- α_3 and α_4 simply restate the edge relations E_1 and E_2 as formulas:

$$\alpha_3 = \bigwedge_{(v_1, v_2) \in E_1} E^1_{v_1, v_2} \wedge \bigwedge_{v_1, v_2 \in V_1, (v_1, v_2) \notin E_1} \neg E^1_{v_1, v_2}$$

$$\alpha_4 = \bigwedge_{(v_1, v_2) \in E_2} E^2_{v_1, v_2} \wedge \bigwedge_{v_1, v_2 \in V_2, (v_1, v_2) \notin E_2} \neg E^2_{v_1, v_2}$$

Reduction (continued)

- Finally, α_5 ensures that if a node a_{i_1} is mapped to b_{j_1} (i.e. if $M_{a_{i_1},b_{j_1}}$ is set to *true*), a node a_{i_2} is mapped to b_{j_2} (i.e. if $M_{a_{i_2},b_{j_2}}$ is set to *true*), and there is an edge $(a_{i_1}, a_{i_2}) \in E_1$ (i.e. if $E^1_{a_{i_1},a_{i_2}}$ is set to *true*), then there must be an edge $(b_{j_1}, b_{j_2}) \in E_2$ (i.e. $E^2_{b_{j_1},b_{j_2}}$ must also evaluate to *true*):

$$\alpha_5 = \bigwedge_{1 \leq i_1, i_2 \leq n, 1 \leq j_1, j_2 \leq m} (\neg E^1_{a_{i_1},a_{i_2}} \vee \neg M_{a_{i_1},b_{j_1}} \vee \neg M_{a_{i_2},b_{j_2}} \vee E^2_{b_{j_1},b_{j_2}})$$

This concludes the definition of the reduction. The reduction clearly works in polynomial time. It remains to prove its correctness, i.e., there is a homomorphism h from G_1 to $G_2 \Leftrightarrow \varphi$ is satisfiable.

Correctness of the Reduction

"$\Rightarrow$" Suppose there is a homomorphism h from G_1 to G_2. We define an assignment T that makes φ evaluate to *true*, and thus show satisfiability of φ. We define the truth assignment T by setting the following variables to *true*:

- $M_{a_i,h(a_i)}$ for each $a_i \in V_1$;
- $E^1_{a_i,a_j}$ for all edges $(a_i, a_j) \in E_1$;
- $E^2_{b_i,b_j}$ for all edges $(b_i, b_j) \in E_2$;

The remaining variables are set to *false*.

It remains to show that φ evaluates to true in T. To this end, we have to show that each of the subformulas $\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5$ evaluates to *true*. This is easy and therefore left as an exercise (inspect each subformula α_i and give an argument why T as defined above satisfies α_i).

Correctness of the Reduction (continued)

"$\Leftarrow$" Suppose φ is satisfiable, i.e. there is a truth assignment T under which all formulas α_i, $1 \leq i \leq 5$, evaluate to *true*. We have to show that there is a homomorphism h from G_1 to G_2.

We define h as follows: For each $a_i \in V_1$, let $h(a_i) = b_j$, where $b_j \in V_2$ is the unique vertex such that M_{a_i,b_j} is assigned *true* by T. The existence of such a unique vertex is guaranteed since α_1 and α_2 evaluate to *true* by assumption. Hence, the mapping h is well-defined.

It remains to show that h is indeed a homomorphism. Let $(a_1, a_2) \in E_1$ be an arbitrary edge of G_1. We must show that $(h(a_1), h(a_2)) \in E_2$. The argument is as follows:

- Since α_3 evaluates to *true*, we have that $E^1_{a_1,a_2}$ is assigned *true* by T.
- By the construction of h, we have that $M_{a_1,h(a_1)}$ and $M_{a_2,h(a_2)}$ are both assigned *true* by T.
- Since α_5 evaluates to *true*, it must be the case then that $E^2_{h(a_1),h(a_2)}$ is assigned *true* by T.
- Since α_4 evaluates to *true*, we must have $(h(a_1), h(a_2)) \in E_2$.

NP-Completeness and Algorithm Design Techniques

Showing that a problem is NP-complete implies that the problem is not in P unless NP = P (which is considered very unlikely).

When a problem is known to be NP-complete, further efforts are usually directed to:

- Heuristics
- Attacking special cases
- Approximation algorithms
- Randomized algorithms
- (Exponential) algorithms that are practical for small instances
- etc.

Learning Objectives

- You should now be familiar with the intuition of NP-completeness (and recognize NP-complete problems).
- Two fundamental NP-complete problems: **SAT** and **3-SAT**.
- Difference between logical equivalence and sat-equivalence.
- Many more examples of NP-complete problems, e.g.: **CLIQUE**, **INDEPENDENT SET**, **VERTEX COVER**, **3-COLORABILITY**, **HAMILTON-PATH**, **HAMILTON-CYCLE**, **TSP(D)**, etc.
- Usefulness of reductions to **SAT**.