NP-Completeness of 3-COLOR

Stepan Kuznetsov

Discrete Math Bridging Course, HSE University

k-COLOR

- A **correct coloring** of an undirected graph $G = (V, E)$ in k colors is a function
  $c \colon V \to C, \qquad$ where $|C| = k$
  (e.g., $C = \{1, \dots, k\}$)
  and for any $(u, v) \in E$ we have $c(u) \neq c(v)$.

k-COLOR

- A **correct coloring** of an undirected graph $G = (V, E)$ in k colors is a function
  $c \colon V \to C, \qquad$ where $|C| = k$
  (e.g., $C = \{1, \ldots, k\}$)
  and for any $(u, v) \in E$ we have $c(u) \neq c(v)$.
- The algorithmic problem k-COLOR:
  determine whether a given graph G is colorable in k colors.

k-COLOR

- A **correct coloring** of an undirected graph $G = (V, E)$ in k colors is a function
  $c \colon V \to C, \qquad$ where $|C| = k$
  (e.g., $C = \{1, \ldots, k\}$)
  and for any $(u, v) \in E$ we have $c(u) \neq c(v)$.
- The algorithmic problem k-COLOR: determine whether a given graph G is colorable in k colors.
- We have

$$k\text{-COLOR} \leq_m^P (k+1)\text{-COLOR},$$

  thus, complexity of k-COLOR grows with k.

k-COLOR

- k-COLOR $\in$ NP for any k.

k-COLOR

- k-COLOR $\in$ NP for any k.
- 1-COLOR is trivial.

k-COLOR

- k-COLOR $\in$ NP for any k.
- 1-COLOR is trivial.
- Since

$$\text{2-COLOR} \leq_m^P \text{2-SAT},$$

  we have 2-COLOR $\in$ P.

k-COLOR

- k-COLOR $\in$ NP for any k.
- 1-COLOR is trivial.
- Since

$$\text{2-COLOR} \leq_m^P \text{2-SAT},$$

  we have 2-COLOR $\in$ P.
- For $k = 3$, we also have

$$\text{3-COLOR} \leq_m^P \text{3-SAT},$$

  but this is just a particular case of Cook–Levin.

k-COLOR

- k-COLOR $\in$ NP for any k.
- 1-COLOR is trivial.
- Since

$$\text{2-COLOR} \leq_m^P \text{2-SAT},$$

  we have 2-COLOR $\in$ P.
- For $k = 3$, we also have

$$\text{3-COLOR} \leq_m^P \text{3-SAT},$$

  but this is just a particular case of Cook–Levin.
- In order to prove NP-hardness of 3-COLOR, we need the **opposite** reduction.

3-COLOR

Theorem

3-SAT $\leq_m^P$ 3-COLOR, and therefore 3-COLOR is NP-complete.

3-COLOR

Theorem

3-SAT $\leq_m^P$ 3-COLOR, and therefore 3-COLOR is NP-complete.

- We need to construct a polynomially computable reducing function

$$f\colon \varphi \mapsto G_\varphi,$$

such that for any 3-CNF Boolean formula φ it is satisfiable if and only if the graph G_φ is colorable in 3 colors.

3-COLOR

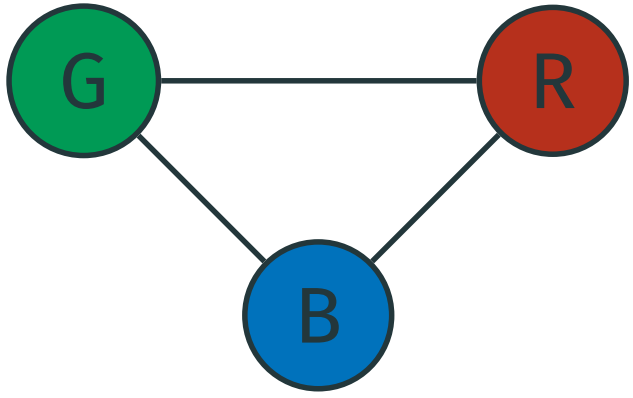
- Let $C = \{R, G, B\}$.

3-COLOR

- Let $C = \{R, G, B\}$.
- As we encode satisfying assignments (of φ) as 3-colorings (of G_φ), red will intuitively mean false, green is true, and blue is the neutral third color.

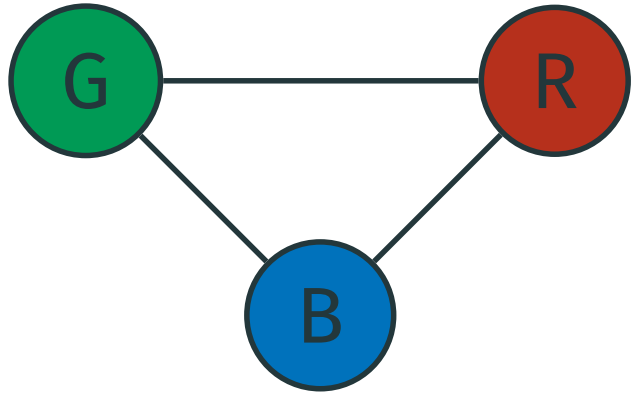
3-COLOR

- Let $C = \{R, G, B\}$.
- As we encode satisfying assignments (of φ) as 3-colorings (of G_φ), red will intuitively mean false, green is true, and blue is the neutral third color.
- We start with a triangle called **palette**:



3-COLOR

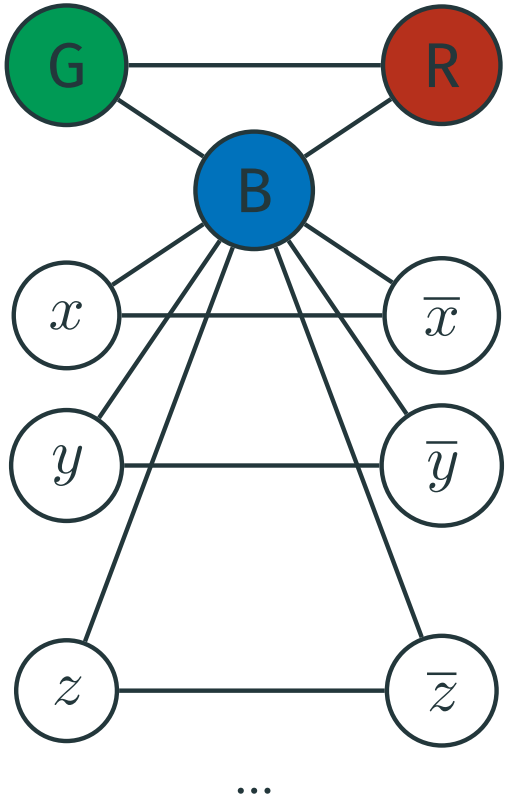
- Let $C = \{R, G, B\}$.
- As we encode satisfying assignments (of φ) as 3-colorings (of G_φ), red will intuitively mean false, green is true, and blue is the neutral third color.
- We start with a triangle called **palette**:



- We may suppose that the palette is colored as shown (otherwise rename the colors).

3-COLOR: Modelling Assignments

Next, we introduce a pair of vertices (x and $\bar{x}$) for each variable, and connect them as shown:



3-COLOR: Modelling Assignments

- For each variable x, the vertices x, $\bar{x}$, and B form a triangle.

3-COLOR: Modelling Assignments

- For each variable x, the vertices x, $\bar{x}$, and B form a triangle.
- Thus,
  either x is green and $\bar{x}$ is red (x is true),
  or x is red and $\bar{x}$ is green (x is false).

3-COLOR: Modelling Assignments

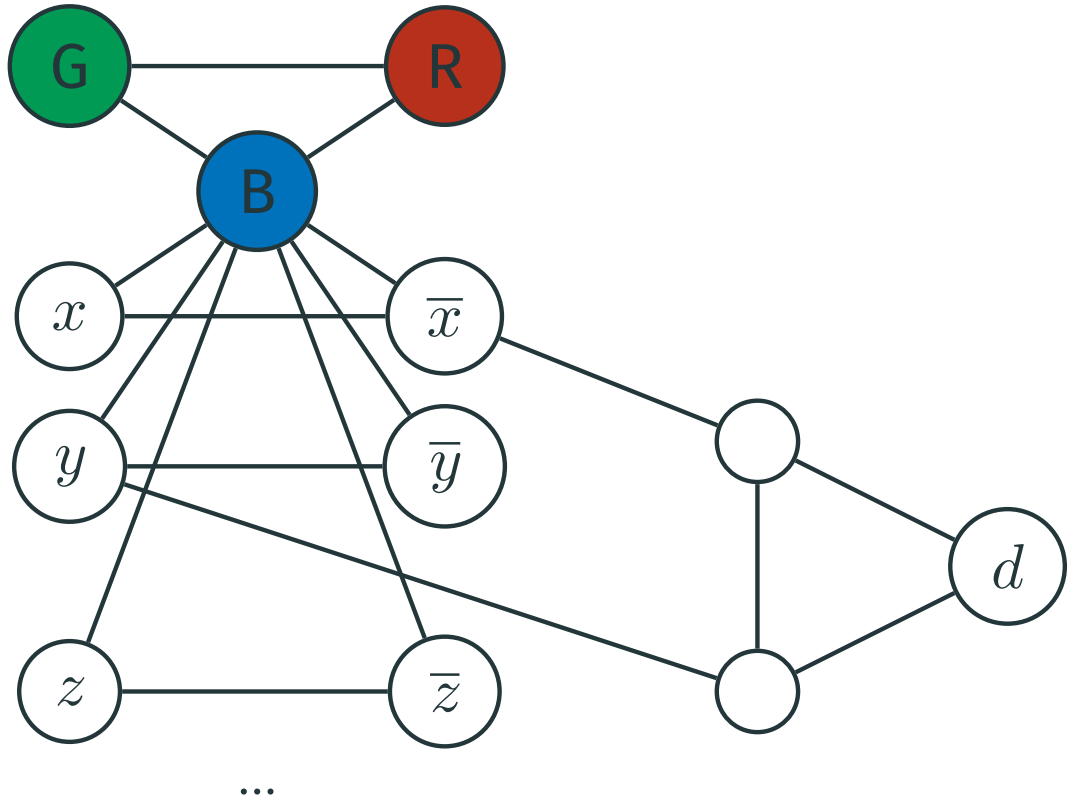
- For each variable x, the vertices x, $\bar{x}$, and B form a triangle.
- Thus,
  either x is green and $\bar{x}$ is red (x is true),
  or x is red and $\bar{x}$ is green (x is false).
- Each correct 3-coloring of the graph we have so far corresponds to a truth assignment.

3-COLOR: Modelling Assignments

- For each variable x, the vertices x, $\bar{x}$, and B form a triangle.
- Thus,
  either x is green and $\bar{x}$ is red (x is true),
  or x is red and $\bar{x}$ is green (x is false).
- Each correct 3-coloring of the graph we have so far corresponds to a truth assignment.
- Now we add **constraints** so that the assignment satisfies the 3-CNF φ.

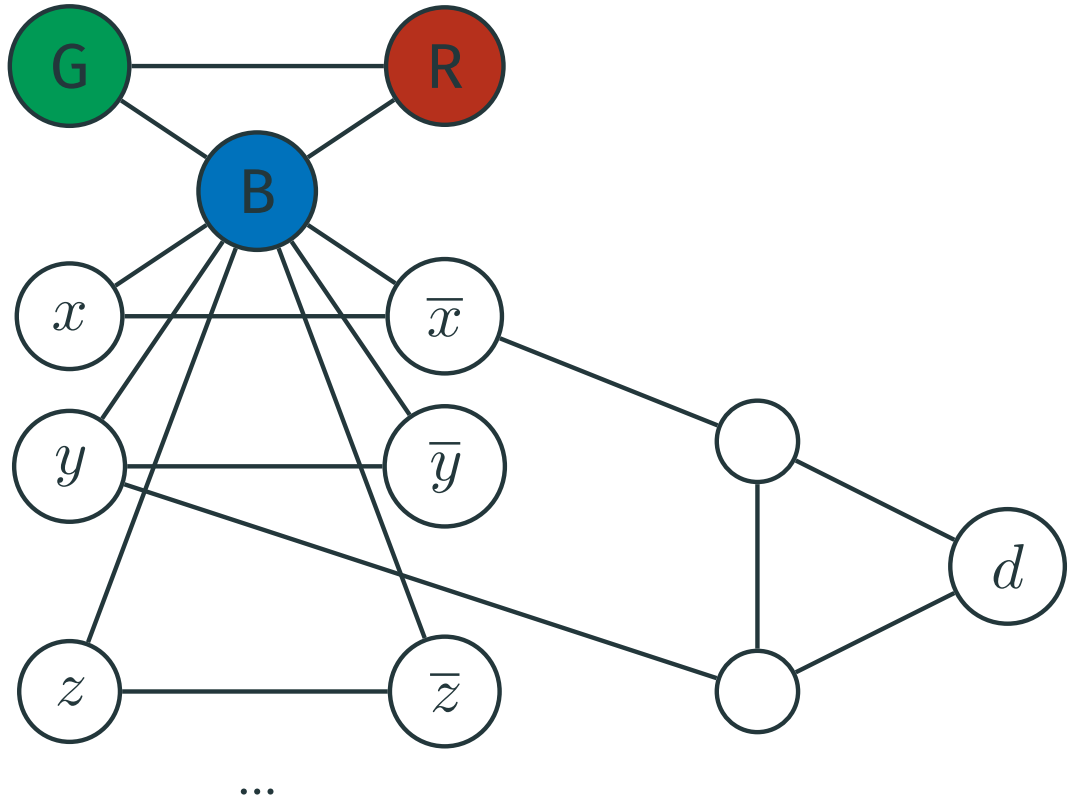
3-COLOR: Satisfying a 2-Clause

We start with modelling a 2-clause, e.g., $\bar{x} \vee y$.



3-COLOR: Satisfying a 2-Clause

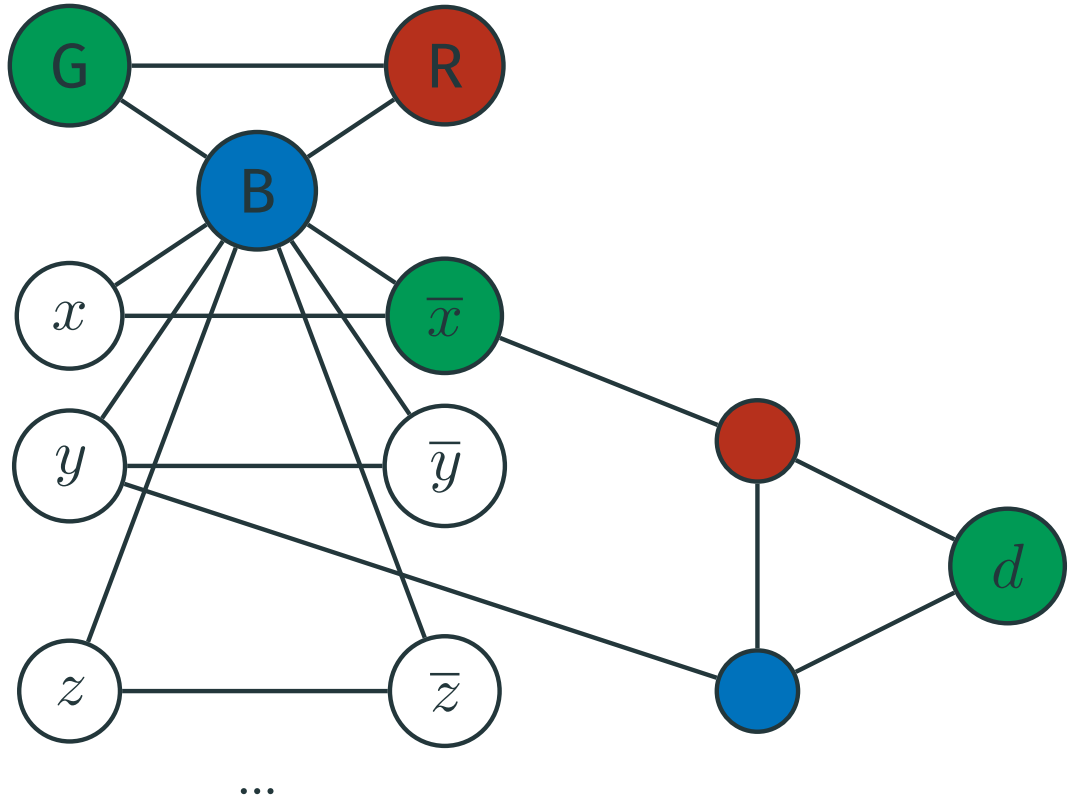
We start with modelling a 2-clause, e.g., $\bar{x} \vee y$.



Claim: Vertex d **can be** colored in green if and only if $\bar{x}$ is green and y is green (maybe both).

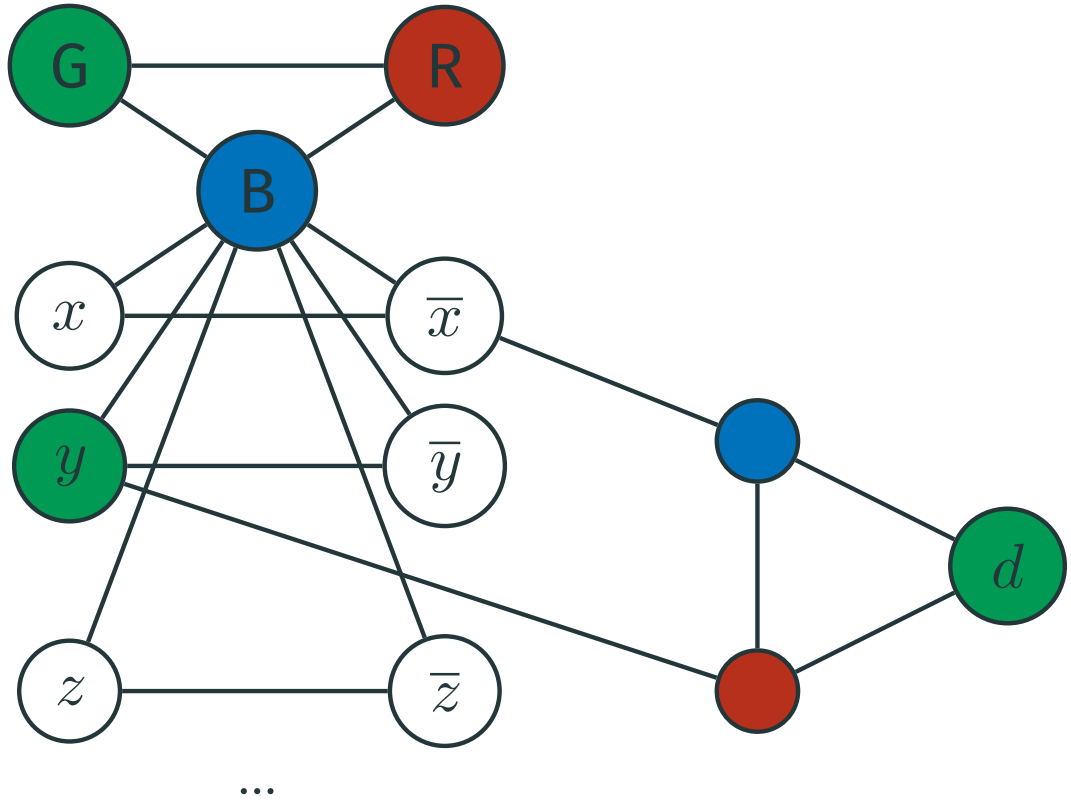
3-COLOR: Satisfying a 2-Clause

If $\bar{x}$ or y is green, then we may color as follows:



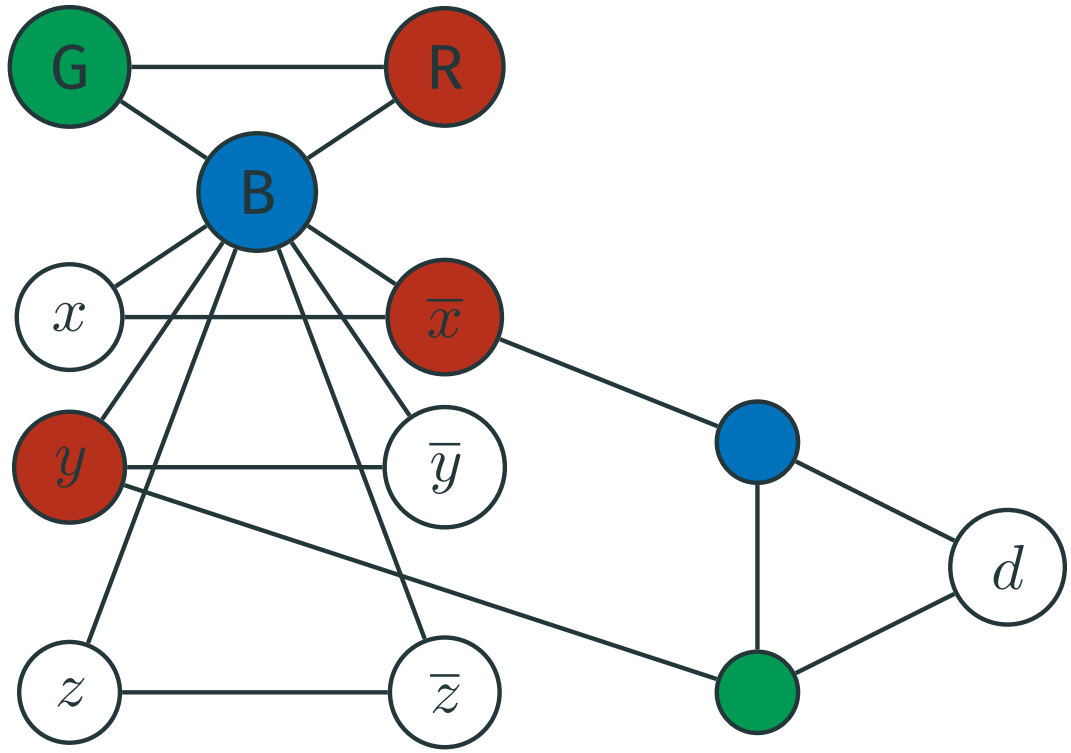
3-COLOR: Satisfying a 2-Clause

If $\bar{x}$ or y is green, then we may color as follows:



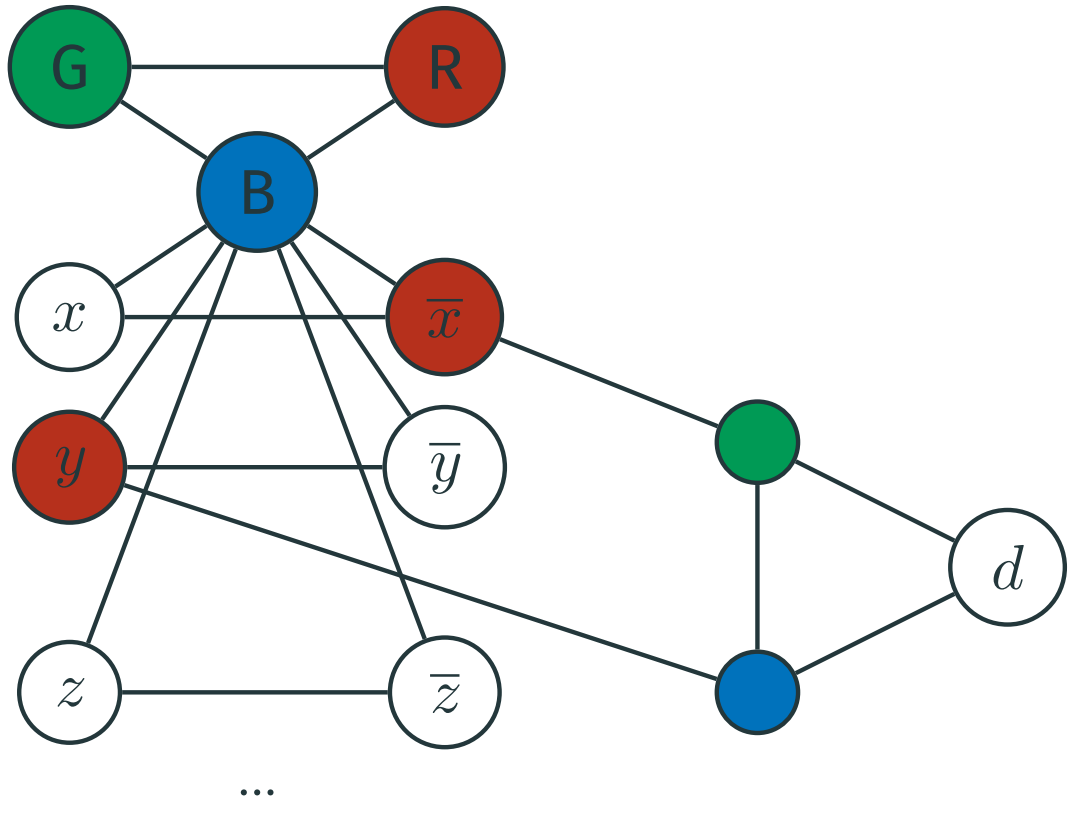
3-COLOR: Satisfying a 2-Clause

If both $\bar{x}$ and y are red, then one of the two unnamed vertices should be green, which prevents d from being green.



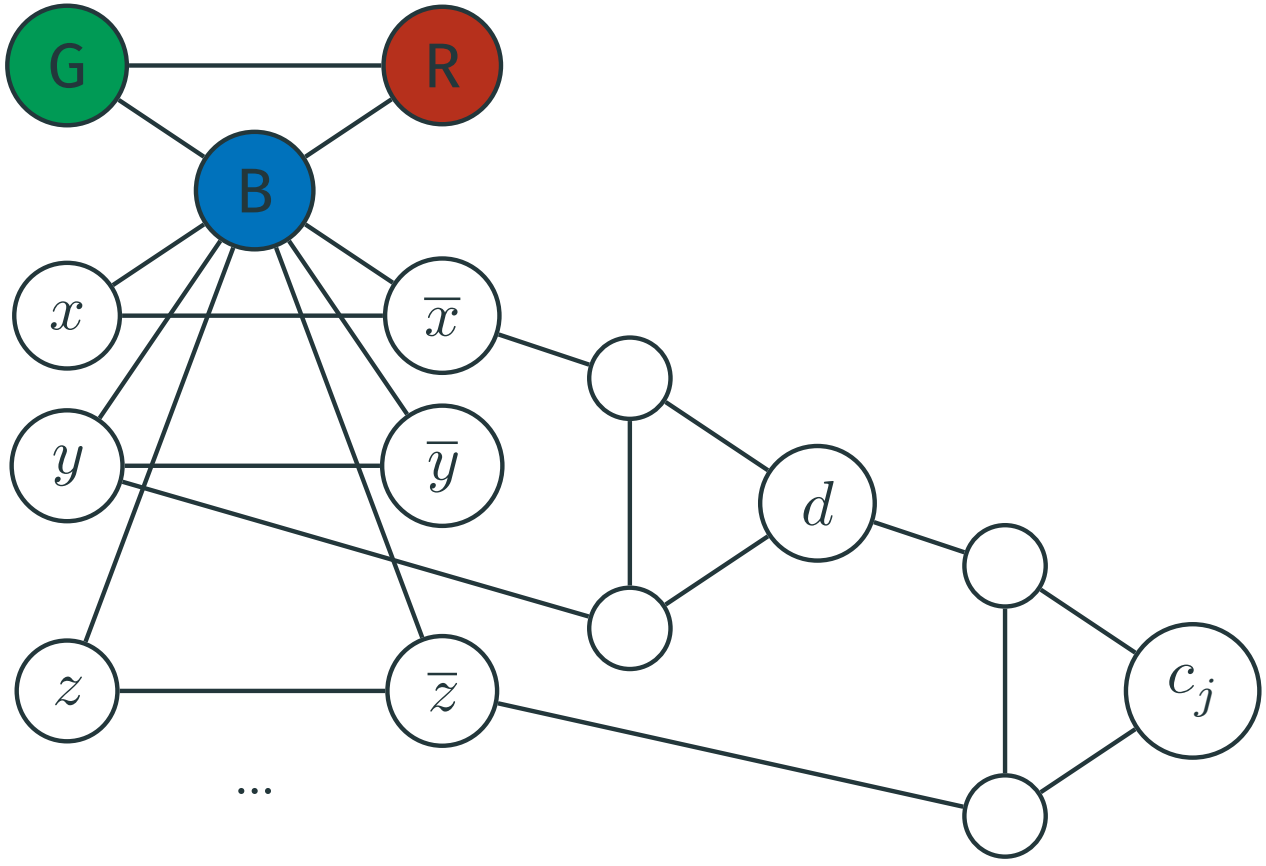
3-COLOR: Satisfying a 2-Clause

If both $\bar{x}$ and y are red, then one of the two unnamed vertices should be green, which prevents d from being green.



3-COLOR: Satisfying a 3-Clause

Now for a 3-clause, e.g., $C_j = \bar{x} \vee y \vee \bar{z}$, we copy the triangle once more:



3-COLOR: Satisfying a 3-Clause

- Here d corresponds to $\bar{x} \vee y$, and can be made green if and only if $\bar{x} \vee y$.

3-COLOR: Satisfying a 3-Clause

- Here d corresponds to $\bar{x} \vee y$, and can be made green if and only if $\bar{x} \vee y$.
- In the same way, c_j corresponds to $C_j = (\bar{x} \vee y) \vee \bar{z}$.

3-COLOR: Satisfying a 3-Clause

- Here d corresponds to $\bar{x} \vee y$, and can be made green if and only if $\bar{x} \vee y$.
- In the same way, c_j corresponds to $C_j = (\bar{x} \vee y) \vee \bar{z}$.
- We introduce such a construction with c_j on top for each 3-clause C_j.

3-COLOR: Satisfying a 3-Clause

- Here d corresponds to $\bar{x} \vee y$, and can be made green if and only if $\bar{x} \vee y$.
- In the same way, c_j corresponds to $C_j = (\bar{x} \vee y) \vee \bar{z}$.
- We introduce such a construction with c_j on top for each 3-clause C_j.
    - For 2-clauses we use just one triangle, and for a 1-clause c_j is just x or $\bar{x}$.

3-COLOR: Satisfying a 3-Clause

- Here d corresponds to $\bar{x} \vee y$, and can be made green if and only if $\bar{x} \vee y$.
- In the same way, c_j corresponds to $C_j = (\bar{x} \vee y) \vee \bar{z}$.
- We introduce such a construction with c_j on top for each 3-clause C_j.
  - For 2-clauses we use just one triangle, and for a 1-clause c_j is just x or $\bar{x}$.
- Finally, we connect each c_j to both R and B, in order to enforce c_j being green.

3-COLOR: Satisfying a 3-Clause

- Here d corresponds to $\bar{x} \vee y$, and can be made green if and only if $\bar{x} \vee y$.
- In the same way, c_j corresponds to $C_j = (\bar{x} \vee y) \vee \bar{z}$.
- We introduce such a construction with c_j on top for each 3-clause C_j.
  - For 2-clauses we use just one triangle, and for a 1-clause c_j is just x or $\bar{x}$.
- Finally, we connect each c_j to both R and B, in order to enforce c_j being green.
- The resulting graph G_φ is 3-colorable if and only if φ is satisfiable.

Course Overview

- This course is a quick intro into the ideas of **discrete mathematics** and **complexity theory**, for the use in data science.

Course Overview

- This course is a quick intro into the ideas of **discrete mathematics** and **complexity theory**, for the use in data science.
- Discrete objects can be encoded as **words** over $\{0, 1\}$ (or, more generally, over a finite alphabet Σ).

Course Overview

- This course is a quick intro into the ideas of **discrete mathematics** and **complexity theory**, for the use in data science.
- Discrete objects can be encoded as **words** over $\{0, 1\}$ (or, more generally, over a finite alphabet Σ).
- An **algorithm** takes such a word as its **input**, and, in the simple case, returns another word as the **output**.

Course Overview

- This course is a quick intro into the ideas of **discrete mathematics** and **complexity theory**, for the use in data science.
- Discrete objects can be encoded as **words** over $\{0, 1\}$ (or, more generally, over a finite alphabet Σ).
- An **algorithm** takes such a word as its **input**, and, in the simple case, returns another word as the **output**.
    - We have also considered more sophisticated behaviour, e.g., algorithms with delays.

Course Overview

- Thus, an algorithm computes a function

$$f\colon \{0,1\}^* \to \{0,1\}^*,$$

and such functions are called **computable**.

Course Overview

- Thus, an algorithm computes a function

$$f\colon \{0,1\}^* \to \{0,1\}^*,$$

  and such functions are called **computable**.
- We also take care of the **running time** of the algorithm.

Course Overview

- Thus, an algorithm computes a function

$$f\colon \{0,1\}^* \to \{0,1\}^*,$$

  and such functions are called **computable**.
- We also take care of the **running time** of the algorithm.
- A reasonable notion of "effectivity" is **polynomial time computation**, where computing $f(x)$ takes $\leq p(|x|)$ steps (where p is a fixed polynomial).

Course Overview

- An **algorithmic problem** is a question whether a given function f is computable, and if it is, whether it is computable effectively.

Course Overview

- An **algorithmic problem** is a question whether a given function f is computable, and if it is, whether it is computable effectively.
- Special case: **decision problem**, where the output is just one bit ($f\colon \{0,1\}^* \to \{0,1\}$).

Course Overview

- An **algorithmic problem** is a question whether a given function f is computable, and if it is, whether it is computable effectively.
- Special case: **decision problem**, where the output is just one bit ($f\colon \{0,1\}^* \to \{0,1\}$).
- Another way is to express a decision problem as a subset (predicate) $A \subseteq \{0,1\}^*$, where $A = \{x \mid f(x) = 1\}$.

Course Overview

- A decision problem belongs to class P, if f is polynomially computable.

Course Overview

- A decision problem belongs to class P, if f is polynomially computable.
- The next higher complexity class is NP ("non-deterministically polynomial").

Course Overview

- A decision problem belongs to class P, if f is polynomially computable.
- The next higher complexity class is NP ("non-deterministically polynomial").
- Def. 1: the problem is in NP, if there is a non-deterministic polynomial-time algorithm, such that $x \in A$ iff the algorithm yields 1 on at least one path ("angelic choice").

Course Overview

- Def. 2:

$$x \in A \iff \exists y\,(|y| \leq p(|x|)\ \&\ R(x,y)),$$

where $R \in \mathrm{P}$.

Course Overview

- Def. 2:

$$x \in A \iff \exists y\,(|y| \leq p(|x|)\ \&\ R(x,y)),$$

  where $R \in \mathrm{P}$.
- In this definition, the statement $x \in A$ is certified by a **witness** y.

Course Overview

- Def. 2:

$$x \in A \iff \exists y\,(|y| \leq p(|x|)\ \&\ R(x,y)),$$

  where $R \in \mathrm{P}$.
- In this definition, the statement $x \in A$ is certified by a **witness** y.
- Any problem from NP is algorithmically solvable (brute-force search for witness).

Course Overview

- Def. 2:

$$x \in A \iff \exists y\,(|y| \leq p(|x|)\ \&\ R(x,y)),$$

  where $R \in \mathrm{P}$.
- In this definition, the statement $x \in A$ is certified by a **witness** y.
- Any problem from NP is algorithmically solvable (brute-force search for witness).
- The model case is **satisfiability of Boolean formulae** (SAT).

Course Overview

- A Boolean formula is built from a set of variables Var and constants 0 and 1 using Boolean operations: $\vee$, $\wedge$, $\neg$, $\rightarrow$.

Course Overview

- A Boolean formula is built from a set of variables Var and constants 0 and 1 using Boolean operations: $\vee$, $\wedge$, $\neg$, $\rightarrow$.
- An **assignment** (truth assignment) is a function $\alpha \colon \mathrm{Var} \to \{0, 1\}$.

Course Overview

- A Boolean formula is built from a set of variables Var and constants 0 and 1 using Boolean operations: $\vee$, $\wedge$, $\neg$, $\rightarrow$.
- An **assignment** (truth assignment) is a function $\alpha \colon \mathrm{Var} \to \{0, 1\}$.
- α is a **satisfying assignment** for formula φ, if the value of φ under α is 1 (true).

Course Overview

- A Boolean formula is built from a set of variables Var and constants 0 and 1 using Boolean operations: $\vee$, $\wedge$, $\neg$, $\rightarrow$.
- An **assignment** (truth assignment) is a function $\alpha \colon \mathrm{Var} \to \{0, 1\}$.
- α is a **satisfying assignment** for formula φ, if the value of φ under α is 1 (true).
- **Checking** this condition is easy (polynomial); **finding** a satisfying assignment for a given φ could be hard.

Course Overview

- Unfortunately, we do not know that P $\neq$ NP.

Course Overview

- Unfortunately, we do not know that $P \neq NP$.
  - It is possible that $P = NP$, then SAT is polynomial.

Course Overview

- Unfortunately, we do not know that $P \neq NP$.
  - It is possible that $P = NP$, then SAT is polynomial.
- We show "hardness" of SAT by a conditional argument, comparing decision problems by their complexity.

Course Overview

- Unfortunately, we do not know that P $\neq$ NP.
  - It is possible that P $=$ NP, then SAT is polynomial.
- We show "hardness" of SAT by a conditional argument, comparing decision problems by their complexity.
- For two problems $A, B \subseteq \{0,1\}^*$, we say

$$A \leq_m^P B \iff \forall x \, (x \in A \iff f(x) \in B)$$

for some polynomially computable function f **(polynomial Carp reduction)**.

Course Overview

- On decision problems (in particular, on the NP class), $\leq_m^P$ is a preorder.

Course Overview

- On decision problems (in particular, on the NP class), $\leq_m^P$ is a preorder.
- The subclass P forms a cluster of $\leq_m^P$-equivalent problems on the bottom of NP.

Course Overview

- On decision problems (in particular, on the NP class), $\leq_m^P$ is a preorder.
- The subclass P forms a cluster of $\leq_m^P$-equivalent problems on the bottom of NP.
  - There are also two degenerate problems, which are even simpler: $A = \emptyset$ and $A = \{0,1\}^*$ (“always no” or “always yes”).

Course Overview

- On decision problems (in particular, on the NP class), $\leq_m^P$ is a preorder.
- The subclass P forms a cluster of $\leq_m^P$-equivalent problems on the bottom of NP.
    - There are also two degenerate problems, which are even simpler: $A = \emptyset$ and $A = \{0,1\}^*$ ("always no" or "always yes").
- On top of NP, there are **NP-complete** problems—the hardest ones, w.r.t. $\leq_m^P$.

Course Overview

- A problem B is **NP-hard**, if for any problem $A \in$ NP we have

$$A \leq_m^P B.$$

Course Overview

- A problem B is **NP-hard**, if for any problem $A \in$ NP we have

$$A \leq_m^P B.$$

- A problem is **NP-complete** if it is NP-hard and belongs to NP.

Course Overview

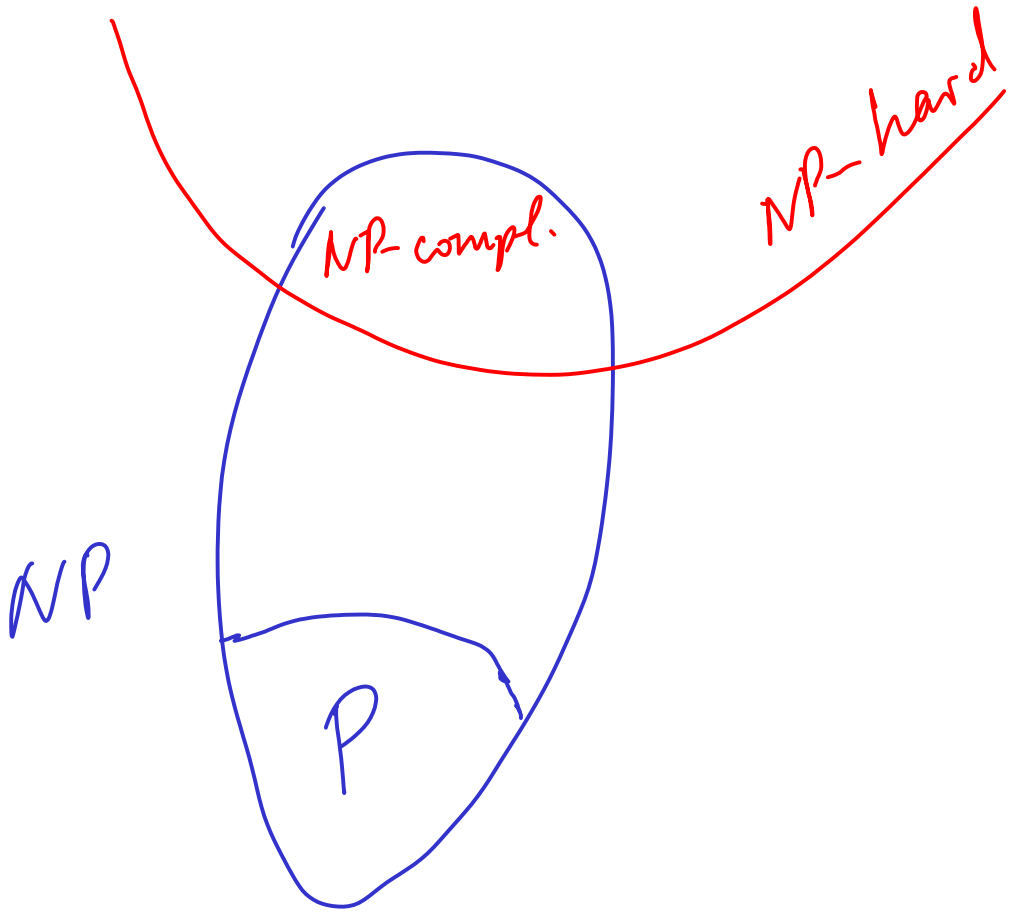
- A problem B is **NP-hard**, if for any problem $A \in$ NP we have

$$A \leq_m^P B.$$

- A problem is **NP-complete** if it is NP-hard and belongs to NP.
  - For example, the validity problem for **first-order** formulae (with quantifiers) is NP-hard, but not NP-complete (being undecidable).

Course Overview

Complexity landscape, if P $\neq$ NP:



Course Overview

- The existence of NP-complete (“maximal”) problems in NP is shown by presenting a concrete example.

Course Overview

- The existence of NP-complete (“maximal”) problems in NP is shown by presenting a concrete example.

Theorem (Cook–Levin)

SAT is NP-complete.

Course Overview

- The existence of NP-complete (“maximal”) problems in NP is shown by presenting a concrete example.

Theorem (Cook–Levin)

SAT is NP-complete.

  - In order to prove $A \leq_m^P$ SAT for an arbitrary $A \in$ NP, we encode the **protocol** of a non-deterministic Turing machine computing A as a Boolean vector, and write a Boolean formula $\varphi_{A,x}$ which states its correctness.

Course Overview

- The existence of NP-complete ("maximal") problems in NP is shown by presenting a concrete example.

Theorem (Cook–Levin)

SAT is NP-complete.

  - In order to prove $A \leq_m^P$ SAT for an arbitrary $A \in$ NP, we encode the **protocol** of a non-deterministic Turing machine computing A as a Boolean vector, and write a Boolean formula $\varphi_{A,x}$ which states its correctness.
  - $x \in A \iff \varphi_{A,x}$ is satisfiable.

Course Overview

- Potentially simpler subproblems of SAT are obtained by considering **special classes** of Boolean formulae.

Course Overview

- Potentially simpler subproblems of SAT are obtained by considering **special classes** of Boolean formulae.
- A **CNF** (conjunctive normal form) is a big conjunction of **clauses** of the form $\neg x \vee y \vee \neg z$.

Course Overview

- Potentially simpler subproblems of SAT are obtained by considering **special classes** of Boolean formulae.
- A **CNF** (conjunctive normal form) is a big conjunction of **clauses** of the form $\neg x \vee y \vee \neg z$.
- In a k-CNF, each clause includes $\leq k$ literals.

Course Overview

- Potentially simpler subproblems of SAT are obtained by considering **special classes** of Boolean formulae.
- A **CNF** (conjunctive normal form) is a big conjunction of **clauses** of the form $\neg x \vee y \vee \neg z$.
- In a k-CNF, each clause includes $\leq k$ literals.
- A **DNF** (disjunctive normal form) is a big disjunction of clauses of the form $\neg x \wedge y \wedge \neg z$.

Course Overview

- Any Boolean formula can be translated into an equivalent CNF or DNF, but this translation is (in general) **not polynomial.**

Course Overview

- Any Boolean formula can be translated into an equivalent CNF or DNF, but this translation is (in general) **not polynomial.**
- DNF-SAT is polynomially solvable: one just has to find at least one non-contradictory clause.

Course Overview

- Any Boolean formula can be translated into an equivalent CNF or DNF, but this translation is (in general) **not polynomial.**
- DNF-SAT is polynomially solvable: one just has to find at least one non-contradictory clause.
- For CNF-SAT, there is the **resolution method**, which is a bit more advanced than brute-force.

Course Overview

- In the resolution method, the CNF is **saturated** by applying the resolution rule:

$$\frac{A \vee p \quad \neg p \vee B}{A \vee B}$$

Course Overview

- In the resolution method, the CNF is **saturated** by applying the resolution rule:

$$\frac{A \vee p \quad \neg p \vee B}{A \vee B}$$

- **Completeness theorem:** the CNF is satisfiable iff it does not include the empty clause (“false”) after saturation.

Course Overview

- In the resolution method, the CNF is **saturated** by applying the resolution rule:

$$\frac{A \vee p \quad \neg p \vee B}{A \vee B}$$

- **Completeness theorem:** the CNF is satisfiable iff it does not include the empty clause ("false") after saturation.
- Unfortunately, for $k \geq 3$ saturation can lead to exponential blowup.

Course Overview

Theorem (Cook–Levin, Tseitin)

3-SAT is NP-complete.

Course Overview

Theorem (Cook–Levin, Tseitin)

3-SAT is NP-complete.

- In contrast, **2-SAT belongs to P**, since applying resolutions to 2-clauses yields also 2-clauses, and there are a polynomial number of such.

Course Overview

Theorem (Cook–Levin, Tseitin)

3-SAT is NP-complete.

- In contrast, **2-SAT belongs to P**, since applying resolutions to 2-clauses yields also 2-clauses, and there are a polynomial number of such.
- Also, resolution allows to solve the **search problem** for 2-SAT (yield at least one satisfying assignment), or yield all satisfying assignments with **polynomial delay**.

Course Overview

- The second half of the course is devoted to **graphs**.

Course Overview

- The second half of the course is devoted to **graphs**.
- Graphs and their generalisations (e.g., hypergraphs) are a generic representation of **structured data** in numerous applications: social networks, bioinformatics, GIS, network topology, etc.

Course Overview

- The second half of the course is devoted to **graphs**.
- Graphs and their generalisations (e.g., hypergraphs) are a generic representation of **structured data** in numerous applications: social networks, bioinformatics, GIS, network topology, etc.
- Graph theory provides many examples of problems from the NP class: "given a graph G, determine whether it includes some specific substructure".

Course Overview

- For some of these problems, we proved that they belong to P: finding an Euler path / cycle; 2-colorability (via reduction to 2-SAT).

Course Overview

- For some of these problems, we proved that they belong to P: finding an Euler path / cycle; 2-colorability (via reduction to 2-SAT).
- For others, we proved NP-completeness. These include 3-COLOR (and k-COLOR for $k \geq 3$); Hamitonian path / cycle search (both directed and undirected); subgraph isomorphism and its special cases (CLIQUE, INDSET, VERTEXCOVER).

Course Overview

- Finally, the graph isomorphism problem is neither known to belong to P, neither known to be NP-complete.
- For proving NP-completeness, we usually construct a reduction **from** 3-SAT, e.g.:

$$\text{3-SAT} \leq_m^P \text{3-COLOR},$$

while for proving polynomial solvability the **opposite** reduction can be used:

$$\text{2-COLOR} \leq_m^P \text{2-SAT}.$$