Circuits, $\mathcal{P}$- and $\mathcal{NP}$-complete problems

Peter Hajnal

Bolyai Institute, SzTE, Szeged

2023 fall

Normalized Turing Machines

Reminder

For $L \in_T \mathcal{TIME}(t(n))/\mathcal{NTIME}(t(n))$, we always assume that $t(n)$ is a nice time function, i.e., there exists a Turing machine that solves L and run for $t(n)$ steps on each input of length n.
For an input $\omega \in \Sigma^n$, the run of T can be represented as

$$\kappa_0(\omega) \to \kappa_1 \to \kappa_2 \to \ldots \to \kappa_\ell,$$

where $\kappa_0(\omega)$ is the initial configuration corresponding to ω, κ_{i+1} is the successor of κ_i, and κ_ℓ is the first configuration where the state is ACCEPT or REJECT. We can assume $\ell = t(n)$.

Encoding Configurations with Bits

Encoding Configurations with Bits

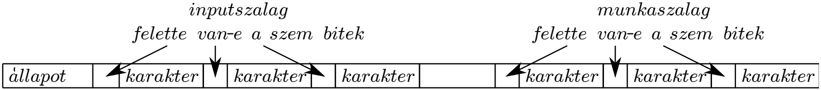
Observation

Configurations κ can be encoded with bit sequences.

Encoding Configurations with Bits

Observation

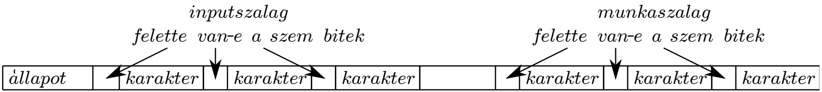
Configurations κ can be encoded with bit sequences.



Encoding Configurations with Bits

Observation

Configurations κ can be encoded with bit sequences.

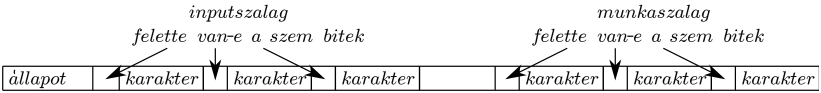


We also need to encode the position of the head.

Encoding Configurations with Bits

Observation

Configurations κ can be encoded with bit sequences.

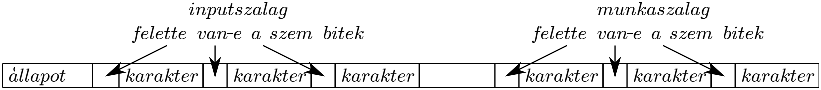


We also need to encode the position of the head. To do this, we *spread* it by placing one bit on each cell.

Encoding Configurations with Bits

Observation

Configurations κ can be encoded with bit sequences.

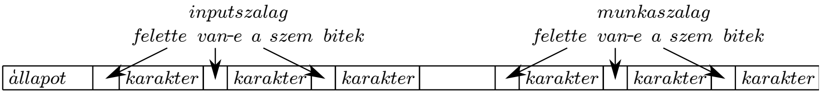


We also need to encode the position of the head. To do this, we *spread* it by placing one bit on each cell. This means that not every bit sequence of a given length encodes a configuration.

Encoding Configurations with Bits

Observation

Configurations κ can be encoded with bit sequences.



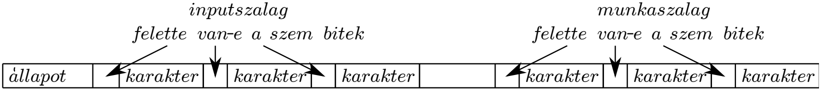
We also need to encode the position of the head. To do this, we *spread* it by placing one bit on each cell. This means that not every bit sequence of a given length encodes a configuration.

Encoding S elements requires $\lceil \log_2 |S| \rceil$ bits.

Encoding Configurations with Bits

Observation

Configurations κ can be encoded with bit sequences.



We also need to encode the position of the head. To do this, we *spread* it by placing one bit on each cell. This means that not every bit sequence of a given length encodes a configuration.

Encoding S elements requires $\lceil \log_2 |S| \rceil$ bits.

The length of blocks encoding states and symbols depends on $|S|$, $|\Sigma|$, and $|\Gamma|$. In any case, a constant number of bits is sufficient (depending on the Turing machine).

Observations

Observations

1st Remark

We can choose the agreement such that for a given n-bit input ω, its length $\lceil\omega\rceil$ is $\alpha_T \cdot n$.

Observations

1st Remark

We can choose the agreement such that for a given n-bit input ω, its length $\lceil\omega\rceil$ is $\alpha_T \cdot n$.

If the time complexity of T is at most $t(n)$, then the length of the configuration codes is $\beta_T \cdot t(n)$.

Observations

1st Remark

We can choose the agreement such that for a given n-bit input ω, its length $\lceil\omega\rceil$ is $\alpha_T \cdot n$.

If the time complexity of T is at most $t(n)$, then the length of the configuration codes is $\beta_T \cdot t(n)$.

In the following, n and the encoding agreement are always fixed (accordingly, the lengths of the corresponding codes are always known).

Further Remark

Further Remark

2nd Remark

The function $\lceil \kappa_i \rceil \to \lceil \kappa_i^+ \rceil = \lceil \kappa_{i+1} \rceil$ can be easily defined/calculated.

Further Remark

2nd Remark

The function $\lceil \kappa_i \rceil \to \lceil \kappa_i^+ \rceil = \lceil \kappa_{i+1} \rceil$ can be easily defined/calculated.

The statement of the remark is not mathematically precise, and the interpretation of the term *easily* is not well-defined.

Further Remark

2nd Remark

The function $\lceil \kappa_i \rceil \to \lceil \kappa_i^+ \rceil = \lceil \kappa_{i+1} \rceil$ can be easily defined/calculated.

The statement of the remark is not mathematically precise, and the interpretation of the term *easily* is not well-defined.

We show that it is possible to straightforwardly determine/calculate a small (polynomial-size) circuit that, given the code of a configuration, computes the code of the next configuration.

Circuits

Circuits

Definition: Circuit

A circuit is an $\overrightarrow{G}$ directed graph that does not contain a directed cycle (i.e., it can be drawn so that all edges go *downwards*).

Circuits

Definition: Circuit

A circuit is an $\overrightarrow{G}$ directed graph that does not contain a directed cycle (i.e., it can be drawn so that all edges go *downwards*).

Each vertex has in-degree $0, 1$, or 2.

Circuits

Definition: Circuit

A circuit is an $\overrightarrow{G}$ directed graph that does not contain a directed cycle (i.e., it can be drawn so that all edges go *downwards*).

Each vertex has in-degree $0, 1$, or 2. Vertices with in-degree 0 are called input vertices. Let I be the set of input vertices.

Circuits

Definition: Circuit

A circuit is an $\overrightarrow{G}$ directed graph that does not contain a directed cycle (i.e., it can be drawn so that all edges go *downwards*).

Each vertex has in-degree $0, 1$, or 2. Vertices with in-degree 0 are called input vertices. Let I be the set of input vertices. We refer to the non-input vertices as gates. The set of gates is $K = V(\overrightarrow{G}) - I$.

Circuits

Definition: Circuit

A circuit is an $\overrightarrow{G}$ directed graph that does not contain a directed cycle (i.e., it can be drawn so that all edges go *downwards*).

Each vertex has in-degree $0, 1$, or 2. Vertices with in-degree 0 are called input vertices. Let I be the set of input vertices. We refer to the non-input vertices as gates. The set of gates is $K = V(\overrightarrow{G}) - I$. A special vertex or vertices, referred to as output vertices, is/are designated for output.

Circuits

Definition: Circuit

A circuit is an $\overrightarrow{G}$ directed graph that does not contain a directed cycle (i.e., it can be drawn so that all edges go *downwards*).

Each vertex has in-degree $0, 1$, or 2. Vertices with in-degree 0 are called input vertices. Let I be the set of input vertices. We refer to the non-input vertices as gates. The set of gates is $K = V(\overrightarrow{G}) - I$. A special vertex or vertices, referred to as output vertices, is/are designated for output. Let $\ell_I : I \to \{x_1, x_2, \ldots, x_n\} \cup \{0, 1\}$ be a labeling for the output vertices.

Circuits

Definition: Circuit

A circuit is an $\overrightarrow{G}$ directed graph that does not contain a directed cycle (i.e., it can be drawn so that all edges go *downwards*).

Each vertex has in-degree $0, 1$, or 2. Vertices with in-degree 0 are called input vertices. Let I be the set of input vertices. We refer to the non-input vertices as gates. The set of gates is $K = V(\overrightarrow{G}) - I$. A special vertex or vertices, referred to as output vertices, is/are designated for output. Let $\ell_I : I \to \{x_1, x_2, \ldots, x_n\} \cup \{0, 1\}$ be a labeling for the output vertices. Let $\ell_K : K \to \{\neg, \vee, \wedge\}$ be a labeling for the gates, with the property that the label of a gate is $\neg$ if its in-degree is 1.

Circuits

Definition: Circuit

A circuit is an $\overrightarrow{G}$ directed graph that does not contain a directed cycle (i.e., it can be drawn so that all edges go *downwards*).

Each vertex has in-degree $0, 1$, or 2. Vertices with in-degree 0 are called input vertices. Let I be the set of input vertices. We refer to the non-input vertices as gates. The set of gates is $K = V(\overrightarrow{G}) - I$. A special vertex or vertices, referred to as output vertices, is/are designated for output. Let $\ell_I : I \to \{x_1, x_2, \ldots, x_n\} \cup \{0, 1\}$ be a labeling for the output vertices. Let $\ell_K : K \to \{\neg, \vee, \wedge\}$ be a labeling for the gates, with the property that the label of a gate is $\neg$ if its in-degree is 1. Let $\ell = (\ell_I, \ell_K)$ be the labeling function for all vertices.

Circuits

Definition: Circuit

A circuit is an $\overrightarrow{G}$ directed graph that does not contain a directed cycle (i.e., it can be drawn so that all edges go *downwards*).

Each vertex has in-degree $0, 1$, or 2. Vertices with in-degree 0 are called input vertices. Let I be the set of input vertices. We refer to the non-input vertices as gates. The set of gates is $K = V(\overrightarrow{G}) - I$. A special vertex or vertices, referred to as output vertices, is/are designated for output. Let $\ell_I : I \to \{x_1, x_2, \ldots, x_n\} \cup \{0, 1\}$ be a labeling for the output vertices. Let $\ell_K : K \to \{\neg, \vee, \wedge\}$ be a labeling for the gates, with the property that the label of a gate is $\neg$ if its in-degree is 1. Let $\ell = (\ell_I, \ell_K)$ be the labeling function for all vertices.

A labeled directed graph $(\overrightarrow{G}, \ell)$ is called a circuit.

Circuit as a Dynamic Computation Model

Circuit as a Dynamic Computation Model

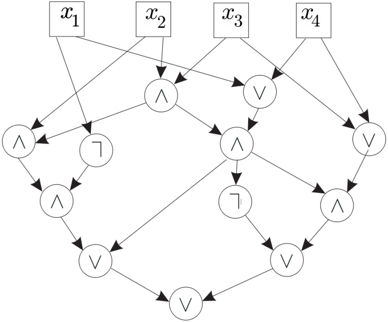
A circuit computes a Boolean function in the following way:

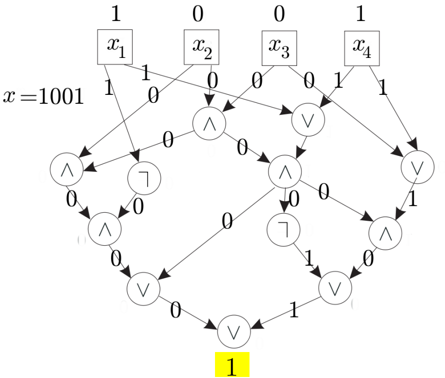
Circuit as a Dynamic Computation Model

A circuit computes a Boolean function in the following way: We assume that the circuit is drawn such that all edges go downward. The evaluation of gates proceeds from top to bottom.

Circuit as a Dynamic Computation Model

A circuit computes a Boolean function in the following way: We assume that the circuit is drawn such that all edges go downward. The evaluation of gates proceeds from top to bottom.





Circuit as a Dynamic Computation Model (continued)

Circuit as a Dynamic Computation Model (continued)

When we reach a gate, the gates from which edges lead to it have already been evaluated, meaning that a computed bit has been assigned.

Circuit as a Dynamic Computation Model (continued)

When we reach a gate, the gates from which edges lead to it have already been evaluated, meaning that a computed bit has been assigned.

The bit computed by the current gate can be naturally interpreted based on the bits flowing on the incoming edges and the label of the gate. The bit sequence computed by the circuit is the bit(s) computed by the output vertex (or vertices).

Circuit as a Dynamic Computation Model (continued)

When we reach a gate, the gates from which edges lead to it have already been evaluated, meaning that a computed bit has been assigned.

The bit computed by the current gate can be naturally interpreted based on the bits flowing on the incoming edges and the label of the gate. The bit sequence computed by the circuit is the bit(s) computed by the output vertex (or vertices).

Definition

Let $f_{\mathcal{C}}$ be the Boolean function computed/realized by the circuit $\mathcal{C}$ as described above.

Mathematizing the Last Observation

Mathematizing the Last Observation

2nd Observation

From a bit sequence encoding a configuration, we can straightforwardly describe a small circuit that computes the code of the next configuration.

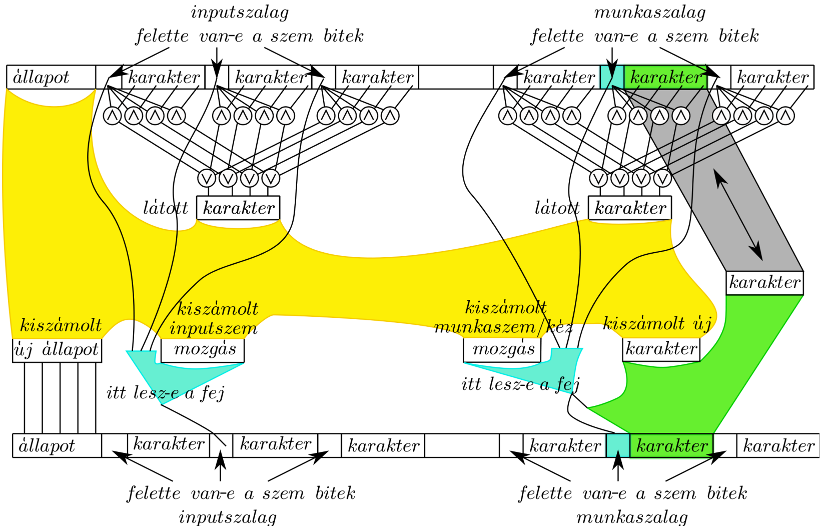
Mathematizing the Last Observation

2nd Observation

From a bit sequence encoding a configuration, we can straightforwardly describe a small circuit that computes the code of the next configuration.

Our construction is simple but involves many details and agreements. Instead of providing a formal description, we illustrate the ideas through an example.

Mathematizing the Last Observation in a Picture



Mathematatizing the Last Observation in Words

Mathematizing the Last Observation in Words

For a cell, we perform the logical AND operation on the bit indicating whether the eye/hand is there and the bits encoding the content of the cell. The resulting bit sequence is either all 0 (if the eye/hand is not there, we ANDed with all 0s) or the code of the seen character (if the eye/hand is there).

Mathematizing the Last Observation in Words

For a cell, we perform the logical AND operation on the bit indicating whether the eye/hand is there and the bits encoding the content of the cell. The resulting bit sequence is either all 0 (if the eye/hand is not there, we ANDed with all 0s) or the code of the seen character (if the eye/hand is there).

For the bit sequences obtained for the tape cells, we read the first, then the second, and so on, characters by performing the logical OR operation. We obtain the code of the seen character on the tape.

Mathematatizing the Last Observation in Words

Mathematizing the Last Observation in Words

In the yellow area, more complex calculations are performed: we compute constant many bits from constant many bits (depending on the Turing machine). The concrete implementation depends on the transition function. If we have no idea about the dependencies of individual bits, we can write down the obvious DNF formula based on the transitions. Even in this case, working with a constant number of gates allows us to accomplish our task.

Mathematizing the Last Observation in Words

In the yellow area, more complex calculations are performed: we compute constant many bits from constant many bits (depending on the Turing machine). The concrete implementation depends on the transition function. If we have no idea about the dependencies of individual bits, we can write down the obvious DNF formula based on the transitions. Even in this case, working with a constant number of gates allows us to accomplish our task.

In the light blue area, we calculate one of the bits describing the position of the head. This depends on whether the head was there or stood over one of the neighboring cells, and also on the direction the transition prescribes. This part could be explicitly written down, but it is unnecessary based on our previous remark. This blue part is there for each head-position bit.

Mathematatizing the Last Observation in Words

Mathematizing the Last Observation in Words

In the green area, we calculate the new content of the tape. Each tape cell has such a green block (we only displayed one for simplicity). The new character depends on the new one, the old one, and whether the head is there. This part could also be easily implemented if we knew the number ℓ of bits used to encode the elements of Γ. In the green area, we calculate the function $f(\epsilon, k_0, k_1) = k_\epsilon$, which computes ℓ from $1 + 2\ell$ bits.

Observation

Observation

3rd Observation

$\lceil \omega \rceil \to \lceil \kappa_0(\omega) \rceil$ is a simple assignment/function.

Observation

3rd Observation

$\lceil \omega \rceil \to \lceil \kappa_0(\omega) \rceil$ is a simple assignment/function.

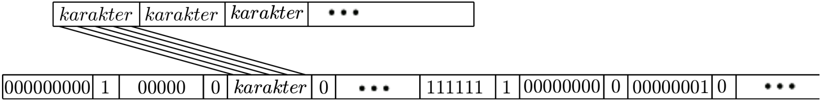
This is a simpler observation than before. Again, we refer to a diagram.

Observation

3rd Observation

$\lceil\omega\rceil \to \lceil\kappa_0(\omega)\rceil$ is a simple assignment/function.

This is a simpler observation than before. Again, we refer to a diagram.

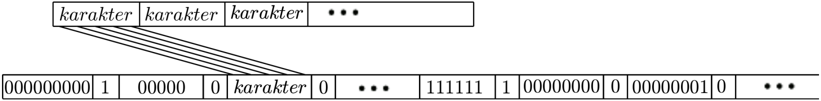


Observation

3rd Observation

$\lceil \omega \rceil \to \lceil \kappa_0(\omega) \rceil$ is a simple assignment/function.

This is a simpler observation than before. Again, we refer to a diagram.



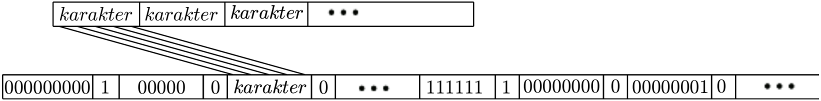
We assumed that the code for the START state is $00\ldots0$ (of length $\lceil \log_2 |S| \rceil$, in our example 9).

Observation

3rd Observation

$\lceil\omega\rceil \to \lceil\kappa_0(\omega)\rceil$ is a simple assignment/function.

This is a simpler observation than before. Again, we refer to a diagram.



We assumed that the code for the START state is $00\ldots0$ (of length $\lceil\log_2|S|\rceil$, in our example 9).

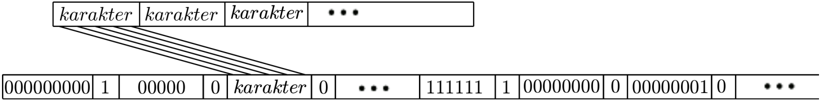
The code for $\triangleright$ on the input tape is $00\ldots0$, and the code for $\triangleleft$ is $11\ldots1$ (of length $\lceil\log_2|\Sigma|\rceil$, in our example 5).

Observation

3rd Observation

$\lceil \omega \rceil \to \lceil \kappa_0(\omega) \rceil$ is a simple assignment/function.

This is a simpler observation than before. Again, we refer to a diagram.



We assumed that the code for the START state is $00\ldots0$ (of length $\lceil \log_2 |S| \rceil$, in our example 9).

The code for $\triangleright$ on the input tape is $00\ldots0$, and the code for $\triangleleft$ is $11\ldots1$ (of length $\lceil \log_2 |\Sigma| \rceil$, in our example 5).

On the tape, the code for $\triangleright$ is $0\ldots00$, and the code for the blank character is $0\ldots01$ (of length $\lceil \log_2 |\Gamma| \rceil$, in our example 8).

Our Results So Far

Our Results So Far

Definition

Let

$$\text{CIRCUIT-EVAL} = \{\lceil \mathcal{C}, \omega \rceil : \mathcal{C}(\omega) = 1\},$$

i.e., the decision problem that, given a circuit $\mathcal{C}$ and a bit sequence ω, decides whether the circuit computes the 1 bit when given the bit values of $x_1, x_2, \ldots$ as ω (i.e., it evaluates $\mathcal{C}_n(\omega)$).

Our Results So Far

Definition

Let

$$\text{CIRCUIT-EVAL} = \{\lceil \mathcal{C}, \omega \rceil : \mathcal{C}(\omega) = 1\},$$

i.e., the decision problem that, given a circuit $\mathcal{C}$ and a bit sequence ω, decides whether the circuit computes the 1 bit when given the bit values of $x_1, x_2, \ldots$ as ω (i.e., it evaluates $\mathcal{C}_n(\omega)$).

Theorem

CIRCUIT-EVAL is $\mathcal{P}$-complete (with respect to $\mathcal{L}$-reductions).

Proof

Proof

Let $L \in_T \mathcal{P}$.

Proof

Let $L \in_T \mathcal{P}$. Let $t(n)$ be a polynomial, and T be the time constraint.

Proof

Let $L \in_T \mathcal{P}$. Let $t(n)$ be a polynomial, and T be the time constraint.

We can assume that T is such that after reaching the ACCEPT/REJECT state, it „holds" its state. Thus, answering the question $\omega \in L$? ($\omega \in \Sigma^n$) is equivalent to determining whether, during the run on ω, the configuration $\kappa_{t(n)}$ in state ACCEPT.

Proof

Let $L \in_T \mathcal{P}$. Let $t(n)$ be a polynomial, and T be the time constraint.

We can assume that T is such that after reaching the ACCEPT/REJECT state, it „holds" its state. Thus, answering the question $\omega \in L$? ($\omega \in \Sigma^n$) is equivalent to determining whether, during the run on ω, the configuration $\kappa_{t(n)}$ in state ACCEPT.

We can agree to encode configurations with 0-1 sequences, such that in the block encoding the state, the code of the ACCEPT state is an all 1s sequence.

Proof

Let $L \in_T \mathcal{P}$. Let $t(n)$ be a polynomial, and T be the time constraint.

We can assume that T is such that after reaching the ACCEPT/REJECT state, it „holds" its state. Thus, answering the question $\omega \in L$? ($\omega \in \Sigma^n$) is equivalent to determining whether, during the run on ω, the configuration $\kappa_{t(n)}$ in state ACCEPT.

We can agree to encode configurations with 0-1 sequences, such that in the block encoding the state, the code of the ACCEPT state is an all 1s sequence.

Based on the above, for any $L \in_T \mathcal{P}$, given an arbitrary $\omega \in \Sigma^n$, we can construct a circuit $\mathcal{T}_{T,\omega}$ that encodes the input gates with ω (3rd observation), and some of its levels encode the elements of the configuration sequence of the Turing computation (2nd observation).

Proof (Continued)

Proof (Continued)

After constructing $t(n)$ such levels, we are interested in determining whether there are only 1s in a specific block. This can be expressed easily using AND gates.

Proof (Continued)

After constructing $t(n)$ such levels, we are interested in determining whether there are only 1s in a specific block. This can be expressed easily using AND gates.

The theoretical part of the reduction follows from the earlier observations. The construction/reduction is log-space.

Circuit Satisfaction and the First $\mathcal{N}P$-Complete Theorem

Circuit Satisfaction and the First $\mathcal{N}P$-Complete Theorem

Definition

$$\text{CIRCUIT-SAT} = \{\lceil \mathcal{C} \rceil : \text{there is a bit sequence } \omega \text{ such that } \mathcal{C}(\omega) = 1\}$$

i.e., the decision problem that, given a circuit $\mathcal{C}$, we need to decide whether it is satisfiable.

Circuit Satisfaction and the First $\mathcal{NP}$-Complete Theorem

Definition

$$\text{CIRCUIT-SAT} = \{\lceil \mathcal{C} \rceil : \text{there is a bit sequence } \omega \text{ such that } \mathcal{C}(\omega) = 1\}$$

i.e., the decision problem that, given a circuit $\mathcal{C}$, we need to decide whether it is satisfiable.

Consequence

(i) For any $L \in \mathcal{NP}$ language, $L \prec_{\mathcal{L}}$ CIRCUIT-SAT

(ii) CIRCUIT-SAT is $\mathcal{NP}$-complete

(iii) $\mathcal{P} = \mathcal{NP} \Leftrightarrow \text{CIRCUIT-SAT} \in \mathcal{P}$

Proof (i)

Proof (i)

$L \in_T \mathcal{NP}$, so for any $\omega \in L$, there exists a witness: $\tau = (t_1, t_2, \ldots, t_{p(n)})$ such that $T(\omega, \tau)$ reaches the ACCEPT state.

Proof (i)

$L \in_T \mathcal{NP}$, so for any $\omega \in L$, there exists a witness: $\tau = (t_1, t_2, \ldots, t_{p(n)})$ such that $T(\omega, \tau)$ reaches the ACCEPT state. That is, for the constructed circuit C, $C(\lceil\omega\rceil, y_1, y_2, \ldots, y_{q(n)})$ evaluates to 1 when we substitute the bits of τ into the corresponding places of the y variables.

Proof (i)

$L \in_T \mathcal{NP}$, so for any $\omega \in L$, there exists a witness: $\tau = (t_1, t_2, \ldots, t_{p(n)})$ such that $T(\omega, \tau)$ reaches the ACCEPT state. That is, for the constructed circuit C, $C(\lceil\omega\rceil, y_1, y_2, \ldots, y_{q(n)})$ evaluates to 1 when we substitute the bits of τ into the corresponding places of the y variables.

Conversely, if we find a satisfying assignment for $C(\lceil\omega\rceil, y_1, y_2, \ldots, y_{q(n)})$, we can find the code of a witness.

Proof (i)

$L \in_T \mathcal{NP}$, so for any $\omega \in L$, there exists a witness: $\tau = (t_1, t_2, \ldots, t_{p(n)})$ such that $T(\omega, \tau)$ reaches the ACCEPT state. That is, for the constructed circuit C, $C(\lceil\omega\rceil, y_1, y_2, \ldots, y_{q(n)})$ evaluates to 1 when we substitute the bits of τ into the corresponding places of the y variables.

Conversely, if we find a satisfying assignment for $C(\lceil\omega\rceil, y_1, y_2, \ldots, y_{q(n)})$, we can find the code of a witness.

Therefore, constructing the code of $C(\lceil\omega\rceil, y_1, y_2, \ldots, y_{q(n)})$ (which can be done in $\mathcal{L}$) is a good reduction.

Proof (ii)+(iii)

Proof (ii)+(iii)

(ii) From part (i) and the fact that CIRCUIT-SAT$\in \mathcal{N}\mathcal{P}$ (witness is a satisfying input), it follows that CIRCUIT-SAT is $\mathcal{N}P$-complete.

Proof (ii)+(iii)

(ii) From part (i) and the fact that CIRCUIT-SAT$\in \mathcal{NP}$ (witness is a satisfying input), it follows that CIRCUIT-SAT is $\mathcal{N}P$-complete.

(iii) Since CIRCUIT-SAT $\in \mathcal{NP}$, if $\mathcal{P} = \mathcal{NP}$, then it is in $\mathcal{P}$ as well.

Proof (ii)+(iii)

(ii) From part (i) and the fact that CIRCUIT-SAT$\in \mathcal{NP}$ (witness is a satisfying input), it follows that CIRCUIT-SAT is $\mathcal{N}P$-complete.

(iii) Since CIRCUIT-SAT $\in \mathcal{NP}$, if $\mathcal{P} = \mathcal{NP}$, then it is in $\mathcal{P}$ as well.

Conversely, if CIRCUIT-SAT$\in \mathcal{P}$, then for any $L \in \mathcal{NP}$ language, we can reduce it to CIRCUIT-SAT, and the reduced problem can be decided in $\mathcal{P}$.

Proof (ii)+(iii)

(ii) From part (i) and the fact that CIRCUIT-SAT$\in \mathcal{NP}$ (witness is a satisfying input), it follows that CIRCUIT-SAT is $\mathcal{N}P$-complete.

(iii) Since CIRCUIT-SAT $\in \mathcal{NP}$, if $\mathcal{P} = \mathcal{NP}$, then it is in $\mathcal{P}$ as well.

Conversely, if CIRCUIT-SAT$\in \mathcal{P}$, then for any $L \in \mathcal{NP}$ language, we can reduce it to CIRCUIT-SAT, and the reduced problem can be decided in $\mathcal{P}$.

The two steps together form a polynomial time algorithm solving the decision problem for L. From this, we obtain $\mathcal{NP} \subseteq \mathcal{P}$, so $\mathcal{P} = \mathcal{NP}$ follows.

Break

Conjunctive Normal Form (CNF)

Conjunctive Normal Form (CNF)

Definition

Let $V = \{x_1, x_2, \ldots, x_n\}$ be a set of variables.

Conjunctive Normal Form (CNF)

Definition

Let $V = \{x_1, x_2, \ldots, x_n\}$ be a set of variables. Let $L = V \dot{\cup} \overline{V}$ be the set of literals ($\overline{V}$ is the set of negated variables, i.e., $\{\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n\}$).

Conjunctive Normal Form (CNF)

Definition

Let $V = \{x_1, x_2, \ldots, x_n\}$ be a set of variables. Let $L = V \dot{\cup} \overline{V}$ be the set of literals ($\overline{V}$ is the set of negated variables, i.e., $\{\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n\}$).

A subset of L is called a clause. In our case, a clause is thought of as the disjunction of the associated literals using the $\vee$ logical operator.

Conjunctive Normal Form (CNF)

Definition

Let $V = \{x_1, x_2, \ldots, x_n\}$ be a set of variables. Let $L = V \dot{\cup} \overline{V}$ be the set of literals ($\overline{V}$ is the set of negated variables, i.e., $\{\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n\}$).

A subset of L is called a clause. In our case, a clause is thought of as the disjunction of the associated literals using the $\vee$ logical operator.

A formula φ in conjunctive normal form (CNF) is a set of clauses.

Conjunctive Normal Form (CNF)

Definition

Let $V = \{x_1, x_2, \ldots, x_n\}$ be a set of variables. Let $L = V\dot{\cup}\overline{V}$ be the set of literals ($\overline{V}$ is the set of negated variables, i.e., $\{\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n\}$).

A subset of L is called a clause. In our case, a clause is thought of as the disjunction of the associated literals using the $\vee$ logical operator.

A formula φ in conjunctive normal form (CNF) is a set of clauses.

For this set of clauses, we think of the clauses as connected by the $\wedge$ logical operator, i.e., the conjunction.

Conjunctive Normal Form (CNF)

Conjunctive Normal Form (CNF)

A CNF formula φ is satisfiable if there is an evaluation of V (which can be naturally extended to an evaluation of L) such that each clause evaluates to true when the corresponding literals are substituted by their evaluated values.

Conjunctive Normal Form (CNF)

A CNF formula φ is satisfiable if there is an evaluation of V (which can be naturally extended to an evaluation of L) such that each clause evaluates to true when the corresponding literals are substituted by their evaluated values.

Definition: The SAT language

$$\text{SAT} = \{\lceil\varphi\rceil : \varphi \text{ is a satisfiable CNF}\}$$

Conjunctive Normal Form (CNF)

A CNF formula φ is satisfiable if there is an evaluation of V (which can be naturally extended to an evaluation of L) such that each clause evaluates to true when the corresponding literals are substituted by their evaluated values.

Definition: The SAT language

$$\text{SAT} = \{\lceil\varphi\rceil : \varphi \text{ is a satisfiable CNF}\}$$

That is, SAT is the problem where, given a CNF formula φ, we need to decide whether it is satisfiable.

The Cook–Levin Theorem

The Cook–Levin Theorem

Cook–Levin Theorem

SAT (satisfiability of CNF formulas) is $\mathcal{NP}$-complete.

From One $\mathcal{N}P$-Complete Problem to Another

From One $\mathcal{N}P$-Complete Problem to Another

It is easy to see that SAT $\in \mathcal{NP}$.

From One $\mathcal{N}P$-Complete Problem to Another

It is easy to see that SAT $\in \mathcal{NP}$.

If we are given a reduction from CIRCUIT-SAT to SAT, then we are done:

From One $\mathcal{N}P$-Complete Problem to Another

It is easy to see that SAT $\in \mathcal{NP}$.

If we are given a reduction from CIRCUIT-SAT to SAT, then we are done: From the previous consequence (i), and the transitivity of polynomial-time reductions, we can reduce any $L \in \mathcal{NP}$ language to SAT.

From One $\mathcal{N}P$-Complete Problem to Another

It is easy to see that SAT $\in \mathcal{NP}$.

If we are given a reduction from CIRCUIT-SAT to SAT, then we are done: From the previous consequence (i), and the transitivity of polynomial-time reductions, we can reduce any $L \in \mathcal{NP}$ language to SAT.

The above two steps are very characteristic once we have our first $\mathcal{N}P$-complete problem.

From One $\mathcal{N}P$-Complete Problem to Another

It is easy to see that SAT $\in \mathcal{NP}$.

If we are given a reduction from CIRCUIT-SAT to SAT, then we are done: From the previous consequence (i), and the transitivity of polynomial-time reductions, we can reduce any $L \in \mathcal{NP}$ language to SAT.

The above two steps are very characteristic once we have our first $\mathcal{N}P$-complete problem.

We don't need the full power of $\mathcal{N}P$ to formulate a problem C. It is enough to formulate a $\mathcal{N}P$-complete problem C' using C.

Plan for the Proof of CIRCUIT-SAT $\preceq$ SAT

Plan for the Proof of CIRCUIT-SAT $\preceq$ SAT

We show that

$$\text{CIRCUIT-SAT} \preceq \text{BOOLEAN-EQUATION-SYSTEM-SAT} \preceq \text{SAT}.$$

Plan for the Proof of CIRCUIT-SAT $\preceq$ SAT

We show that

CIRCUIT-SAT $\preceq$ BOOLEAN-EQUATION-SYSTEM-SAT $\preceq$ SAT.

Definition: Boolean Equation System

An equation system $\varphi_i(x_1, x_2, \ldots, x_n) = \psi_i(x_1, x_2, \ldots, x_n)$, $i = 1, 2, \ldots, \ell$, is called a Boolean equation system if φ_i and ψ_i are Boolean formulas. BOOLEAN-EQUATION-SYSTEM-SAT is the language containing the encodings of solvable/satisfiable Boolean equation systems.

Plan for the Proof of CIRCUIT-SAT $\preceq$ SAT

We show that

$$\text{CIRCUIT-SAT} \preceq \text{BOOLEAN-EQUATION-SYSTEM-SAT} \preceq \text{SAT}.$$

Definition: Boolean Equation System

An equation system $\varphi_i(x_1, x_2, \ldots, x_n) = \psi_i(x_1, x_2, \ldots, x_n)$, $i = 1, 2, \ldots, \ell$, is called a Boolean equation system if φ_i and ψ_i are Boolean formulas. BOOLEAN-EQUATION-SYSTEM-SAT is the language containing the encodings of solvable/satisfiable Boolean equation systems.

Definition

BOOLEAN-EQUATION-SYSTEM-SAT is the language containing the encodings of solvable/satisfiable Boolean equation systems.

From Circuit to Equation System

From Circuit to Equation System

Let H be a circuit.

From Circuit to Equation System

Let H be a circuit. We identify each vertex with a variable.

From Circuit to Equation System

Let H be a circuit. We identify each vertex with a variable. For input vertices, this is the label of the vertex.

From Circuit to Equation System

Let H be a circuit. We identify each vertex with a variable. For input vertices, this is the label of the vertex. For the remaining vertices (gates), we assign distinct variables.

From Circuit to Equation System

Let H be a circuit. We identify each vertex with a variable. For input vertices, this is the label of the vertex. For the remaining vertices (gates), we assign distinct variables.

For each gate, we have an equation:

From Circuit to Equation System

Let H be a circuit. We identify each vertex with a variable. For input vertices, this is the label of the vertex. For the remaining vertices (gates), we assign distinct variables.

For each gate, we have an equation:

> $x_g = \neg x_h$ if g is a negation gate and receives input from gate h.

From Circuit to Equation System

Let H be a circuit. We identify each vertex with a variable. For input vertices, this is the label of the vertex. For the remaining vertices (gates), we assign distinct variables.

For each gate, we have an equation:

$x_g = \neg x_h$ if g is a negation gate and receives input from gate h.

$x_g = x_h \wedge x_{h'}$ if g is an AND gate and receives inputs from gates h and h'.

From Circuit to Equation System

Let H be a circuit. We identify each vertex with a variable. For input vertices, this is the label of the vertex. For the remaining vertices (gates), we assign distinct variables.

For each gate, we have an equation:

$x_g = \neg x_h$ if g is a negation gate and receives input from gate h.

$x_g = x_h \wedge x_{h'}$ if g is an AND gate and receives inputs from gates h and h'.

$x_g = x_h \vee x_{h'}$ if g is an OR gate and receives inputs from gates h and h'.

From Circuit to Equation System

Let H be a circuit. We identify each vertex with a variable. For input vertices, this is the label of the vertex. For the remaining vertices (gates), we assign distinct variables.

For each gate, we have an equation:

$x_g = \neg x_h$ if g is a negation gate and receives input from gate h.

$x_g = x_h \wedge x_{h'}$ if g is an AND gate and receives inputs from gates h and h'.

$x_g = x_h \vee x_{h'}$ if g is an OR gate and receives inputs from gates h and h'.

$x_g = 1$ if g is the output gate of the circuit.

CIRCUIT-SAT $\preceq$ BOOLEAN-EQUATION-SYSTEM-SAT $\preceq$ Proof

CIRCUIT-SAT $\preceq$ BOOLEAN-EQUATION-SYSTEM-SAT $\preceq$ Proof

This gives us a Boolean equation system from the circuit.

CIRCUIT-SAT $\preceq$ BOOLEAN-EQUATION-SYSTEM-SAT $\preceq$ Proof

This gives us a Boolean equation system from the circuit.

If the circuit computes 1 for an assignment (proving the satisfiability), then we obtain a solution for the equation system together with the bits computed by the gates.

CIRCUIT-SAT $\preceq$ BOOLEAN-EQUATION-SYSTEM-SAT $\preceq$ Proof

This gives us a Boolean equation system from the circuit.

If the circuit computes 1 for an assignment (proving the satisfiability), then we obtain a solution for the equation system together with the bits computed by the gates.

Conversely, if we find a solution for the equation system, we can extract the assignment of the original input variables.

CIRCUIT-SAT $\preceq$ BOOLEAN-EQUATION-SYSTEM-SAT $\preceq$ Proof

This gives us a Boolean equation system from the circuit.

If the circuit computes 1 for an assignment (proving the satisfiability), then we obtain a solution for the equation system together with the bits computed by the gates.

Conversely, if we find a solution for the equation system, we can extract the assignment of the original input variables.

Thus, generating the code of $x_g = 1$ (which can be done in $\mathcal{L}$) is a good reduction.

BOOLEAN-EQUATION-SYSTEM-SAT $\preceq$ *SAT* Proof

BOOLEAN-EQUATION-SYSTEM-SAT $\preceq$ *SAT* Proof

Replacing '$=$' with '$\leftrightarrow$' logical symbols in the equations, we obtain the formulas corresponding to each equation.

BOOLEAN-EQUATION-SYSTEM-SAT $\preceq$ *SAT* Proof

Replacing '$=$' with '$\leftrightarrow$' logical symbols in the equations, we obtain the formulas corresponding to each equation.

An assignment satisfies an equation if and only if it makes the associated formula true.

BOOLEAN-EQUATION-SYSTEM-SAT $\preceq$ *SAT* Proof

Replacing '$=$' with '$\leftrightarrow$' logical symbols in the equations, we obtain the formulas corresponding to each equation.

An assignment satisfies an equation if and only if it makes the associated formula true.

Each resulting logical expression has at most three variables. It is easy to convert them to CNF.

BOOLEAN-EQUATION-SYSTEM-SAT $\preceq$ *SAT* Proof

Replacing '$=$' with '$\leftrightarrow$' logical symbols in the equations, we obtain the formulas corresponding to each equation.

An assignment satisfies an equation if and only if it makes the associated formula true.

Each resulting logical expression has at most three variables. It is easy to convert them to CNF.

If we connect the CNF expressions obtained from all equations with the 'and' logical symbol, we also obtain a CNF form.

BOOLEAN-EQUATION-SYSTEM-SAT $\preceq$ *SAT* Proof

Replacing '$=$' with '$\leftrightarrow$' logical symbols in the equations, we obtain the formulas corresponding to each equation.

An assignment satisfies an equation if and only if it makes the associated formula true.

Each resulting logical expression has at most three variables. It is easy to convert them to CNF.

If we connect the CNF expressions obtained from all equations with the 'and' logical symbol, we also obtain a CNF form.

This CNF formula is associated with the equation system.

BOOLEAN-EQUATION-SYSTEM-SAT $\preceq$ *SAT* Proof

Replacing '$=$' with '$\leftrightarrow$' logical symbols in the equations, we obtain the formulas corresponding to each equation.

An assignment satisfies an equation if and only if it makes the associated formula true.

Each resulting logical expression has at most three variables. It is easy to convert them to CNF.

If we connect the CNF expressions obtained from all equations with the 'and' logical symbol, we also obtain a CNF form.

This CNF formula is associated with the equation system.

The assignment of variables can be done in polynomial time.

BOOLEAN-EQUATION-SYSTEM-SAT $\preceq$ *SAT* Proof

Replacing '$=$' with '$\leftrightarrow$' logical symbols in the equations, we obtain the formulas corresponding to each equation.

An assignment satisfies an equation if and only if it makes the associated formula true.

Each resulting logical expression has at most three variables. It is easy to convert them to CNF.

If we connect the CNF expressions obtained from all equations with the 'and' logical symbol, we also obtain a CNF form.

This CNF formula is associated with the equation system.

The assignment of variables can be done in polynomial time.

The solvability of the equation system is equivalent to the satisfiability of the formula.

Where Are We, Where Are We Heading?

Where Are We, Where Are We Heading?

So far, we have seen complete problems for various complexity classes ($\mathcal{NL}, \mathcal{P}, \mathcal{NP}$).

Where Are We, Where Are We Heading?

So far, we have seen complete problems for various complexity classes ($\mathcal{NL}, \mathcal{P}, \mathcal{NP}$).

The case of the $\mathcal{NP}$ class is different.

Where Are We, Where Are We Heading?

So far, we have seen complete problems for various complexity classes ($\mathcal{NL}, \mathcal{P}, \mathcal{NP}$).

The case of the $\mathcal{NP}$ class is different. For many seemingly unrelated, important problems, it turned out that they are $\mathcal{NP}$-complete.

Where Are We, Where Are We Heading?

So far, we have seen complete problems for various complexity classes ($\mathcal{NL}, \mathcal{P}, \mathcal{NP}$).

The case of the $\mathcal{NP}$ class is different. For many seemingly unrelated, important problems, it turned out that they are $\mathcal{NP}$-complete. The class is particularly important because if the suspected $\mathcal{P} \neq \mathcal{NP}$ holds, then there is no polynomial time algorithm for these problems.

Where Are We, Where Are We Heading?

So far, we have seen complete problems for various complexity classes ($\mathcal{NL}, \mathcal{P}, \mathcal{NP}$).

The case of the $\mathcal{NP}$ class is different. For many seemingly unrelated, important problems, it turned out that they are $\mathcal{NP}$-complete. The class is particularly important because if the

suspected $\mathcal{P} \neq \mathcal{NP}$ holds, then there is no polynomial time algorithm for these problems. In other words, we consider them inherently difficult based on theoretical conjectures.

CNF with 3-sized clauses, with clauses at most 3-sized

CNF with 3-sized clauses, with clauses at most 3-sized

Let $(=3)$-SAT be the set of encodings of satisfiable CNF formulas, where each clause contains exactly 3 literals. Let (≤ 3)-SAT be the set of encodings of satisfiable CNF formulas, where each clause contains at most 3 literals.

CNF with 3-sized clauses, with clauses at most 3-sized

Let $(=3)$-SAT be the set of encodings of satisfiable CNF formulas, where each clause contains exactly 3 literals. Let (≤ 3)-SAT be the set of encodings of satisfiable CNF formulas, where each clause contains at most 3 literals.

Lemma

$(=3)$-SAT$\preceq_{\mathcal{P}}$ (≤ 3)-SAT, and (≤ 3)-SAT$\preceq_{\mathcal{P}}$ $(=3)$-SAT.

Proof of the Lemma

Proof of the Lemma

The first reduction is obvious, as $(=3)$-SAT is a special case of (≤ 3)-SAT.

Proof of the Lemma

The first reduction is obvious, as $(=3)$-SAT is a special case of (≤ 3)-SAT.

Conversely, let φ be a (≤ 3)-SAT input.

Proof of the Lemma

The first reduction is obvious, as $(=3)$-SAT is a special case of (≤ 3)-SAT.

Conversely, let φ be a (≤ 3)-SAT input. We transform it into an equivalent formula such that each clause has exactly three literals.

Proof of the Lemma

The first reduction is obvious, as $(=3)$-SAT is a special case of (≤ 3)-SAT.

Conversely, let φ be a (≤ 3)-SAT input. We transform it into an equivalent formula such that each clause has exactly three literals.

Keep the 3-sized clauses of φ, and for each small clause, perform the following operation (in parallel):

Proof of the Lemma

The first reduction is obvious, as $(= 3)$-SAT is a special case of (≤ 3)-SAT.

Conversely, let φ be a (≤ 3)-SAT input. We transform it into an equivalent formula such that each clause has exactly three literals.

Keep the 3-sized clauses of φ, and for each small clause, perform the following operation (in parallel): For a clause $C : \langle \ell_1, \ell_2 \rangle$, replace it with the clause pair $\langle \ell_1, \ell_2, u \rangle$, $\langle \ell_1, \ell_2, \overline{u} \rangle$.

Proof of the Lemma

The first reduction is obvious, as $(= 3)$-SAT is a special case of (≤ 3)-SAT.

Conversely, let φ be a (≤ 3)-SAT input. We transform it into an equivalent formula such that each clause has exactly three literals.

Keep the 3-sized clauses of φ, and for each small clause, perform the following operation (in parallel): For a clause $C : \langle \ell_1, \ell_2 \rangle$, replace it with the clause pair $\langle \ell_1, \ell_2, u \rangle$, $\langle \ell_1, \ell_2, \overline{u} \rangle$.

Repeat this for every small clause. What we obtain is an equivalent 3-CNF.

Proof of the Lemma

The first reduction is obvious, as $(=3)$-SAT is a special case of (≤ 3)-SAT.

Conversely, let φ be a (≤ 3)-SAT input. We transform it into an equivalent formula such that each clause has exactly three literals.

Keep the 3-sized clauses of φ, and for each small clause, perform the following operation (in parallel): For a clause $C : \langle \ell_1, \ell_2 \rangle$, replace it with the clause pair $\langle \ell_1, \ell_2, u \rangle$, $\langle \ell_1, \ell_2, \overline{u} \rangle$.

Repeat this for every small clause. What we obtain is an equivalent 3-CNF.

In the example above, the small clause had two literals. Our idea can be applied to clauses with fewer literals. The result: an equivalent formula with one larger clauses. We need to iterate our idea.

Equivalence

Equivalence

Definition

If two languages are polynomial-time reducible to each other, we say that they are equivalent (with respect to polynomial reduction).

Equivalence

Definition

If two languages are polynomial-time reducible to each other, we say that they are equivalent (with respect to polynomial reduction).

In this case, the two problems are considered equivalent.

Equivalence

Definition

If two languages are polynomial-time reducible to each other, we say that they are equivalent (with respect to polynomial reduction).

In this case, the two problems are considered equivalent. According to the theorem, they are equivalent.

Equivalence

Definition

If two languages are polynomial-time reducible to each other, we say that they are equivalent (with respect to polynomial reduction).

In this case, the two problems are considered equivalent. According to the theorem, they are equivalent.

When we mention the language 3-SAT, we may refer to either of the two languages mentioned above.

Equivalence

Definition

If two languages are polynomial-time reducible to each other, we say that they are equivalent (with respect to polynomial reduction).

In this case, the two problems are considered equivalent. According to the theorem, they are equivalent.

When we mention the language 3-SAT, we may refer to either of the two languages mentioned above. Of course, when we reduce to 3-SAT, we assume that the input CNF is such that each clause has exactly three literals.

Equivalence

Definition

If two languages are polynomial-time reducible to each other, we say that they are equivalent (with respect to polynomial reduction).

In this case, the two problems are considered equivalent. According to the theorem, they are equivalent.

When we mention the language 3-SAT, we may refer to either of the two languages mentioned above. Of course, when we reduce to 3-SAT, we assume that the input CNF is such that each clause has exactly three literals. When reducing from 3-SAT, it doesn't matter if the reduction algorithm produces clauses smaller than three.

3-SAT

3-SAT

Theorem

3-SAT is $\mathcal{NP}$-complete.

3-SAT

Theorem

3-SAT is $\mathcal{NP}$-complete.

3-SAT is trivially in $\mathcal{NP}$ (it is a special case of SAT).

Reduction from SAT to 3-SAT

Reduction from SAT to 3-SAT

Reminder: What Are We Claiming?

For a SAT $\rightarrow$ 3-SAT reduction, we need a function that can be computed in polynomial time, such that $\mathcal{C} \in$ SAT $\Leftrightarrow \mathcal{C}' \in$ 3-SAT.

Reduction from SAT to 3-SAT

Reminder: What Are We Claiming?

For a SAT $\to$ 3-SAT reduction, we need a function that can be computed in polynomial time, such that $\mathcal{C} \in$ SAT $\Leftrightarrow \mathcal{C}' \in$ 3-SAT.

The assignment is as follows: for $C = \langle \ell_1, \dots, \ell_k \rangle$, introduce new variables $u_1, \dots, u_{k-1}$, and add the following clauses to $\mathcal{C}'$:

$$\langle \ell_1, \overline{u}_1 \rangle, \langle u_1, \ell_2, \overline{u}_2 \rangle, \dots, \langle u_{i-1}, \ell_i, \overline{u}_i \rangle, \dots, \langle u_{k-2}, \ell_{k-1}, \overline{u}_{k-1} \rangle, \langle u_{k-1}, \ell_k \rangle$$

Reduction from SAT to 3-SAT

Reminder: What Are We Claiming?

For a SAT $\rightarrow$ 3-SAT reduction, we need a function that can be computed in polynomial time, such that $\mathcal{C} \in$ SAT $\Leftrightarrow \mathcal{C}' \in$ 3-SAT.

The assignment is as follows: for $C = \langle \ell_1, \ldots, \ell_k \rangle$, introduce new variables $u_1, \ldots, u_{k-1}$, and add the following clauses to $\mathcal{C}'$:

$$\langle \ell_1, \overline{u}_1 \rangle, \langle u_1, \ell_2, \overline{u}_2 \rangle, \ldots, \langle u_{i-1}, \ell_i, \overline{u}_i \rangle, \ldots, \langle u_{k-2}, \ell_{k-1}, \overline{u}_{k-1} \rangle, \langle u_{k-1}, \ell_k \rangle$$

Do this for every clause in $\mathcal{C}$. What we get is a 3-CNF.

Proof

Proof

We need to establish the following:

Proof

We need to establish the following:

Proof

We need to establish the following:

(i) $\mathcal{C}'$ can be computed in polynomial time (in the length of the code of $\mathcal{C}$). This is obvious.

Proof

We need to establish the following:

(i) $\mathcal{C}'$ can be computed in polynomial time (in the length of the code of $\mathcal{C}$). This is obvious.

(ii) $\mathcal{C}$ is satisfiable if and only if $\mathcal{C}'$ is.

Proof

We need to establish the following:

(i) $\mathcal{C}'$ can be computed in polynomial time (in the length of the code of $\mathcal{C}$). This is obvious.

(ii) $\mathcal{C}$ is satisfiable if and only if $\mathcal{C}'$ is.

If $\mathcal{C}$ is satisfiable, consider a satisfying assignment.

Proof

We need to establish the following:

(i) $\mathcal{C}'$ can be computed in polynomial time (in the length of the code of $\mathcal{C}$). This is obvious.

(ii) $\mathcal{C}$ is satisfiable if and only if $\mathcal{C}'$ is.

If $\mathcal{C}$ is satisfiable, consider a satisfying assignment. For a clause $C = \langle \ell_1, \ldots, \ell_k \rangle$, let ℓ_i be the first true literal in the clause.

Proof

We need to establish the following:

(i) $\mathcal{C}'$ can be computed in polynomial time (in the length of the code of $\mathcal{C}$). This is obvious.

(ii) $\mathcal{C}$ is satisfiable if and only if $\mathcal{C}'$ is.

If $\mathcal{C}$ is satisfiable, consider a satisfying assignment. For a clause $C = \langle \ell_1, \dots, \ell_k \rangle$, let ℓ_i be the first true literal in the clause.

For C, introduce new variables $u_1, \dots, u_{k-1}$ and add the following clauses to $\mathcal{C}'$:

$$\langle \ell_1, \overline{u}_1 \rangle, \langle u_1, \ell_2, \overline{u}_2 \rangle, \dots, \langle u_{i-1}, \ell_i, \overline{u}_i \rangle, \dots, \langle u_{k-2}, \ell_{k-1}, \overline{u}_{k-1} \rangle, \langle u_{k-1}, \ell_k \rangle$$

Proof

We need to establish the following:

(i) $\mathcal{C}'$ can be computed in polynomial time (in the length of the code of $\mathcal{C}$). This is obvious.

(ii) $\mathcal{C}$ is satisfiable if and only if $\mathcal{C}'$ is.

If $\mathcal{C}$ is satisfiable, consider a satisfying assignment. For a clause $C = \langle \ell_1, \ldots, \ell_k \rangle$, let ℓ_i be the first true literal in the clause.

For C, introduce new variables $u_1, \ldots, u_{k-1}$ and add the following clauses to $\mathcal{C}'$:

$$\langle \ell_1, \overline{u}_1 \rangle, \langle u_1, \ell_2, \overline{u}_2 \rangle, \ldots, \langle u_{i-1}, \ell_i, \overline{u}_i \rangle, \ldots, \langle u_{k-2}, \ell_{k-1}, \overline{u}_{k-1} \rangle, \langle u_{k-1}, \ell_k \rangle$$

Repeat this for every clause in $\mathcal{C}$. What we obtain is a 3-CNF.

Proof, The Other Direction

Proof, The Other Direction

$\mathcal{C}'$ has no satisfying assignment where, for some $\mathcal{C}$ clause $C = \langle \ell_1, \ldots, \ell_k \rangle$, all literals $\ell_1 = \ldots = \ell_k = h$. Because the clause

$$\langle \overline{u}_1 \rangle, \langle u_1, \overline{u}_2 \rangle, \ldots, \langle u_{i-1}, \overline{u}_i \rangle, \ldots, \langle u_{k-2}, \overline{u}_{k-1} \rangle \langle u_{k-1} \rangle$$

is unsatisfiable.

k-SAT

k-SAT

Definition: k-SAT

Let k-SAT be the problem defined as follows: given a conjunctive normal form where each clause has at most k literals (k-CNF), is it satisfiable?

k-SAT

Definition: k-SAT

Let k-SAT be the problem defined as follows: given a conjunctive normal form where each clause has at most k literals (k-CNF), is it satisfiable?

Note

The following reduction chain is evident:

$$2\text{-}SAT \preceq_{\mathcal{P}} 3\text{-}SAT \preceq_{\mathcal{P}} 4\text{-}SAT \preceq_{\mathcal{P}} \ldots \preceq_{\mathcal{P}} k\text{-}SAT \preceq_{\mathcal{P}} \ldots \preceq_{\mathcal{P}} SAT.$$

k-SAT

Definition: k-SAT

Let k-SAT be the problem defined as follows: given a conjunctive normal form where each clause has at most k literals (k-CNF), is it satisfiable?

Note

The following reduction chain is evident:

$$2\text{-}SAT \preceq_{\mathcal{P}} 3\text{-}SAT \preceq_{\mathcal{P}} 4\text{-}SAT \preceq_{\mathcal{P}} \ldots \preceq_{\mathcal{P}} k\text{-}SAT \preceq_{\mathcal{P}} \ldots \preceq_{\mathcal{P}} SAT.$$

It is easy to see that 2-SAT $\in \mathcal{P}$. Moreover, 2-SAT $\in co\mathcal{NL}$.

NOT-ALL-TRUE-SAT

NOT-ALL-TRUE-SAT

Definition

An assignment makes a clause homogeneous if every literal in the clause receives the same truth value. In other words, a clause becomes non-homogeneous if it is satisfied (contains a true literal) but not all literals are true.

Let

$$\text{NOT-ALL-TRUE-SAT} = \{\lceil\varphi\rceil : \varphi \text{ is a CNF that is satisfiable but has no clause where all literals are true}\}$$

The Theorem

The Theorem

Theorem

NOT-ALL-TRUE-SAT is $\mathcal{NP}$-complete.

The Theorem

Theorem

NOT-ALL-TRUE-SAT is $\mathcal{NP}$-complete.

Trivially, NOT-ALL-TRUE-SAT $\in \mathcal{NP}$.

Reduction from SAT to NOT-ALL-TRUE-SAT

Reduction from SAT to NOT-ALL-TRUE-SAT

Reminder: What Are We Claiming?

For an SAT $\rightarrow$ NOT-ALL-TRUE-SAT reduction, we need a function that can be computed in polynomial time, such that $\mathcal{C} \in$ SAT $\Leftrightarrow \mathcal{C}' \in$ NOT-ALL-TRUE-SAT.

Reduction from SAT to NOT-ALL-TRUE-SAT

Reminder: What Are We Claiming?

For an SAT $\rightarrow$ NOT-ALL-TRUE-SAT reduction, we need a function that can be computed in polynomial time, such that $\mathcal{C} \in$ SAT $\Leftrightarrow \mathcal{C}' \in$ NOT-ALL-TRUE-SAT.

The assignment is as follows: for $C = \langle \ell_1, \ldots, \ell_k \rangle$, introduce a new variables s, and add the following clauses to $\mathcal{C}'$:

$$\langle \ell_1, \ldots, \ell_k, s \rangle.$$

Reduction from SAT to NOT-ALL-TRUE-SAT

Reminder: What Are We Claiming?

For an SAT $\rightarrow$ NOT-ALL-TRUE-SAT reduction, we need a function that can be computed in polynomial time, such that $\mathcal{C} \in$ SAT $\Leftrightarrow \mathcal{C}' \in$ NOT-ALL-TRUE-SAT.

The assignment is as follows: for $C = \langle \ell_1, \ldots, \ell_k \rangle$, introduce a new variables s, and add the following clauses to $\mathcal{C}'$:

$$\langle \ell_1, \ldots, \ell_k, s \rangle.$$

Do this for every clause in $\mathcal{C}$. What we get is a NOT-ALL-TRUE-SAT instance.

The Theorem

The Theorem

Theorem

NOT-ALL-TRUE-SAT is $\mathcal{NP}$-complete.

The Theorem

Theorem

NOT-ALL-TRUE-SAT is $\mathcal{NP}$-complete.

NOT-ALL-TRUE-SAT $\in \mathcal{NP}$ is trivial.

The Theorem

Theorem

NOT-ALL-TRUE-SAT is $\mathcal{NP}$-complete.

NOT-ALL-TRUE-SAT $\in \mathcal{NP}$ is trivial.

To complete the proof is an easy exercise.

NOT-ALL-TRUE-3-SAT

NOT-ALL-TRUE-3-SAT

Let NOT-ALL-TRUE-3-SAT be the set of CNFs in which every clause contains at most three literals and there is an assignment of truth values to variables such that every clause in φ is not homogeneous.

NOT-ALL-TRUE-3-SAT

Let NOT-ALL-TRUE-3-SAT be the set of CNFs in which every clause contains at most three literals and there is an assignment of truth values to variables such that every clause in φ is not homogeneous.

Consequence

NOT-ALL-TRUE-3-SAT is $\mathcal{NP}$-complete.

NOT-ALL-TRUE-3-SAT

Let NOT-ALL-TRUE-3-SAT be the set of CNFs in which every clause contains at most three literals and there is an assignment of truth values to variables such that every clause in φ is not homogeneous.

Consequence

NOT-ALL-TRUE-3-SAT is $\mathcal{NP}$-complete.

The proof will be a reduction from NOT-ALL-TRUE-SAT to NOT-ALL-TRUE-3-SAT.

Reduction from NOT-ALL-TRUE-SAT to NOT-ALL-TRUE-3-SAT

Reduction from NOT-ALL-TRUE-SAT to NOT-ALL-TRUE-3-SAT

Let's replicate the SAT $\preceq$ 3-SAT reduction.

Reduction from NOT-ALL-TRUE-SAT to NOT-ALL-TRUE-3-SAT

Let's replicate the SAT $\preceq$ 3-SAT reduction.

If we apply the previous reduction from NOT-ALL-TRUE-SAT φ, we obtain a CNF formula $R(\varphi)$ in which every clause contains at most three literals.

Reduction from NOT-ALL-TRUE-SAT to NOT-ALL-TRUE-3-SAT

Let's replicate the SAT $\preceq$ 3-SAT reduction.

If we apply the previous reduction from NOT-ALL-TRUE-SAT φ, we obtain a CNF formula $R(\varphi)$ in which every clause contains at most three literals.

Assume $\varphi \in$ NOT-ALL-TRUE-SAT, meaning that for some assignment of truth values to variables, each clause contains both true and false literals.

Reduction from NOT-ALL-TRUE-SAT to NOT-ALL-TRUE-3-SAT

Let's replicate the SAT $\preceq$ 3-SAT reduction.

If we apply the previous reduction from NOT-ALL-TRUE-SAT φ, we obtain a CNF formula $R(\varphi)$ in which every clause contains at most three literals.

Assume $\varphi \in$ NOT-ALL-TRUE-SAT, meaning that for some assignment of truth values to variables, each clause contains both true and false literals.

Let C be a clause in φ such that ℓ_i is true, and ℓ_j is false. We can assume $i < j$.

Reduction from NOT-ALL-TRUE-SAT to NOT-ALL-TRUE-3-SAT

Let's replicate the SAT $\preceq$ 3-SAT reduction.

If we apply the previous reduction from NOT-ALL-TRUE-SAT φ, we obtain a CNF formula $R(\varphi)$ in which every clause contains at most three literals.

Assume $\varphi \in$ NOT-ALL-TRUE-SAT, meaning that for some assignment of truth values to variables, each clause contains both true and false literals.

Let C be a clause in φ such that ℓ_i is true, and ℓ_j is false. We can assume $i < j$.

Examine what the reduction constructs from C:

$\langle \ell_1, \overline{u}_1 \rangle, \langle u_1, \ell_2, \overline{u}_2 \rangle, \ldots, \langle u_{i-1}, \ell_i, \overline{u}_i \rangle, \ldots, \langle u_{k-2}, \ell_{k-1}, \overline{u}_{k-1} \rangle, \langle u_{k-1}, \ell_k \rangle.$

Reduction from NOT-ALL-TRUE-SAT to NOT-ALL-TRUE-3-SAT

Let's replicate the SAT $\preceq$ 3-SAT reduction.

If we apply the previous reduction from NOT-ALL-TRUE-SAT φ, we obtain a CNF formula $R(\varphi)$ in which every clause contains at most three literals.

Assume $\varphi \in$ NOT-ALL-TRUE-SAT, meaning that for some assignment of truth values to variables, each clause contains both true and false literals.

Let C be a clause in φ such that ℓ_i is true, and ℓ_j is false. We can assume $i < j$.

Examine what the reduction constructs from C:

$$\langle \ell_1, \overline{u}_1 \rangle, \langle u_1, \ell_2, \overline{u}_2 \rangle, \ldots, \langle u_{i-1}, \ell_i, \overline{u}_i \rangle, \ldots, \langle u_{k-2}, \ell_{k-1}, \overline{u}_{k-1} \rangle, \langle u_{k-1}, \ell_k \rangle.$$

Keep the values of the original variables; below, we describe how the new variables get their values.

NOT-ALL-TRUE-SAT $\preceq$ NOT-ALL-TRUE-3-SAT (continued)

NOT-ALL-TRUE-SAT $\preceq$ NOT-ALL-TRUE-3-SAT (continued)

After ℓ_i, the negation of a new variable is introduced. Assigning a true value to the new variable makes the ℓ_i literal false and true in the *small* clause. Moving right, assigning true values to the subsequent new variables reaches the *small* clause for ℓ_j. Meanwhile, every clause receives both true and false values.

NOT-ALL-TRUE-SAT $\preceq$ NOT-ALL-TRUE-3-SAT (continued)

After ℓ_i, the negation of a new variable is introduced. Assigning a true value to the new variable makes the ℓ_i literal false and true in the *small* clause. Moving right, assigning true values to the subsequent new variables reaches the *small* clause for ℓ_j. Meanwhile, every clause receives both true and false values.

The same can be achieved for the outermost clauses by moving from the edges towards the center and distributing the values of the new variables accordingly.

NOT-ALL-TRUE-SAT $\preceq$ NOT-ALL-TRUE-3-SAT (continued)

After ℓ_i, the negation of a new variable is introduced. Assigning a true value to the new variable makes the ℓ_i literal false and true in the *small* clause. Moving right, assigning true values to the subsequent new variables reaches the *small* clause for ℓ_j. Meanwhile, every clause receives both true and false values.

The same can be achieved for the outermost clauses by moving from the edges towards the center and distributing the values of the new variables accordingly.

In summary, we have shown that

$$R(\varphi) \in \text{NOT-ALL-TRUE-3-SAT}.$$

NOT-ALL-TRUE-SAT $\preceq$ NOT-ALL-TRUE-3-SAT (continued)

NOT-ALL-TRUE-SAT $\preceq$ NOT-ALL-TRUE-3-SAT (continued)

Conversely, assume that $R(\varphi) \in$ NOT-ALL-TRUE-3-SAT. We have seen that an assignment satisfying every clause in $R(\varphi)$ (which is a non-all-true assignment) cannot result in the original clauses having all true values.

NOT-ALL-TRUE-SAT $\preceq$ NOT-ALL-TRUE-3-SAT (continued)

Conversely, assume that $R(\varphi) \in$ NOT-ALL-TRUE-3-SAT. We have seen that an assignment satisfying every clause in $R(\varphi)$ (which is a non-all-true assignment) cannot result in the original clauses having all true values.

We only need to rule out the possibility that $R(\varphi)$ is a non-all-true assignment to the original variables, restricting each clause of the original ones to have every literal true.

NOT-ALL-TRUE-SAT $\preceq$ NOT-ALL-TRUE-3-SAT (continued)

Conversely, assume that $R(\varphi) \in$ NOT-ALL-TRUE-3-SAT. We have seen that an assignment satisfying every clause in $R(\varphi)$ (which is a non-all-true assignment) cannot result in the original clauses having all true values.

We only need to rule out the possibility that $R(\varphi)$ is a non-all-true assignment to the original variables, restricting each clause of the original ones to have every literal true.

This would imply that the

$$\langle \overline{u}_1 \rangle, \langle u_1, \overline{u}_2 \rangle, \ldots, \langle u_{i-1}, \overline{u}_i \rangle, \ldots, \langle u_{k-2}, \overline{u}_{k-1} \rangle \langle u_{k-1} \rangle$$

clauses all need to be false. This (as before) is impossible.

Break

Coloring Problems

Coloring Problems

Definition

k-COLORABILITY is the following problem: given a graph, can it be colored with k colors?

Coloring Problems

Definition

k-COLORABILITY is the following problem: given a graph, can it be colored with k colors?

Theorem

3-COLORABILITY is $\mathcal{NP}$-complete.

Coloring Problems

Definition

k-COLORABILITY is the following problem: given a graph, can it be colored with k colors?

Theorem

3-COLORABILITY is $\mathcal{NP}$-complete.

3-COLORABILITY $\in \mathcal{NP}$: the witness is a coloring, and it can be checked in polynomial time whether it is a valid coloring.

3-SAT $\preceq$ 3-COLORABILITY Reduction

3-SAT $\preceq$ 3-COLORABILITY Reduction

To each 3-CNF $\mathcal{C}$, we assign a graph $G_{\mathcal{C}}$, with vertices n, h, the variables of $\mathcal{C}$, their negations, and for each $C \in \mathcal{C}$ clause, the vertices C_1, C_2, C_3, C_4, C_5.

3-SAT $\preceq$ 3-COLORABILITY Reduction

To each 3-CNF $\mathcal{C}$, we assign a graph $G_{\mathcal{C}}$, with vertices n, h, the variables of $\mathcal{C}$, their negations, and for each $C \in \mathcal{C}$ clause, the vertices C_1, C_2, C_3, C_4, C_5. If $\mathcal{C}$ has n variables and m clauses each containing 3 literals, then the number of vertices in G is

3-SAT $\preceq$ 3-COLORABILITY Reduction

To each 3-CNF $\mathcal{C}$, we assign a graph $G_{\mathcal{C}}$, with vertices n, h, the variables of $\mathcal{C}$, their negations, and for each $C \in \mathcal{C}$ clause, the vertices C_1, C_2, C_3, C_4, C_5. If $\mathcal{C}$ has n variables and m clauses each containing 3 literals, then the number of vertices in G is $2 + 2n + 5m$.

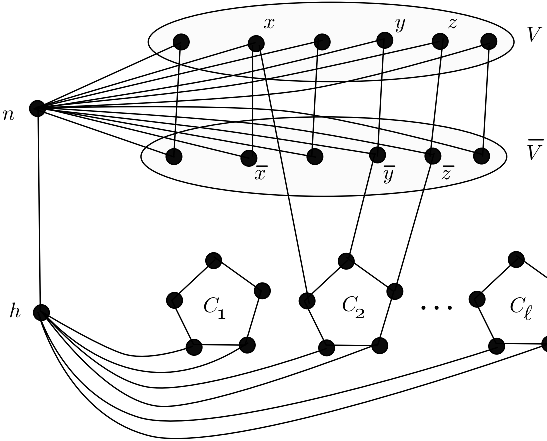
3-SAT $\preceq$ 3-COLORABILITY Reduction

To each 3-CNF $\mathcal{C}$, we assign a graph $G_{\mathcal{C}}$, with vertices n, h, the variables of $\mathcal{C}$, their negations, and for each $C \in \mathcal{C}$ clause, the vertices C_1, C_2, C_3, C_4, C_5. If $\mathcal{C}$ has n variables and m clauses each containing 3 literals, then the number of vertices in G is $2 + 2n + 5m$.

The edges of $G_{\mathcal{C}}$ are as follows: nh, for each variable x_i $x_i\overline{x_i}$, nx_i and $n\overline{x_i}$, and for each $C = \langle z_1, z_2, z_3 \rangle$ clause C_1C_2, C_2C_3, C_3C_4, C_4C_5, C_5C_1, C_1z_1, C_2z_2, C_3z_3, C_4h, C_5h.

3-SAT $\preceq$ 3-COLORABILITY Reduction (Visual)

3-SAT $\preceq$ 3-COLORABILITY Reduction (Visual)



3-SAT $\preceq$ 3-COLORABILITY Reduction (Verbal)

3-SAT $\preceq$ 3-COLORABILITY Reduction (Verbal)

It is easy to verify that $G_{\mathcal{C}}$ can be determined in polynomial time, and it is 3-colorable if and only if $\mathcal{C}$ is satisfiable (using the observation that the 3-coloring of z_1, z_2, z_3, h can be extended to a valid coloring for the 5 vertices corresponding to the clause $C = \langle z_1, z_2, z_3 \rangle$, if the colors of the 4 vertices are distinct).

Additional Coloring Problems

Additional Coloring Problems

Remark

It can be easily checked that 2-COLORABILITY is in $co\mathcal{NL}$.

Additional Coloring Problems

Remark

It can be easily checked that 2-COLORABILITY is in $co\mathcal{NL}$.

Remark (Appel, Haken 1977)

4-COLORABILITY is trivial.

Additional Coloring Problems

Remark

It can be easily checked that 2-COLORABILITY is in $co\mathcal{NL}$.

Remark (Appel, Haken 1977)

4-COLORABILITY is trivial.

Definition

COLORING PROBLEM: given a graph G and a natural number k. Is there a proper k-coloring of G?

Theorem

COLORING PROBLEM is $\mathcal{NP}$-complete.

Proof

Proof

COLORING PROBLEM $\in \mathcal{NP}$: the witness is a coloring, and it can be checked in polynomial time whether it is a proper coloring.

Proof

COLORING PROBLEM $\in \mathcal{NP}$: the witness is a coloring, and it can be checked in polynomial time whether it is a proper coloring.

COLORING PROBLEM is $\mathcal{NP}$-hard, as it generalizes 3-COLORABILITY.

INDEPENDENT-VERTEX-SET

INDEPENDENT-VERTEX-SET

Definition

INDEPENDENT-VERTEX-SET: given a graph G and a natural number k. Does G have an independent set of size k?

INDEPENDENT-VERTEX-SET

Definition

INDEPENDENT-VERTEX-SET: given a graph G and a natural number k. Does G have an independent set of size k?

Theorem

INDEPENDENT-VERTEX-SET is $\mathcal{NP}$-complete.

INDEPENDENT-VERTEX-SET

Definition

INDEPENDENT-VERTEX-SET: given a graph G and a natural number k. Does G have an independent set of size k?

Theorem

INDEPENDENT-VERTEX-SET is $\mathcal{NP}$-complete.

INDEPENDENT-VERTEX-SET $\in \mathcal{NP}$: the witness is an independent set.

Proof I

Proof I

Reduction from SAT:
$\mathcal{C} = (C_1 = \langle z_{1,1}, \ldots, z_{1,r_1}\rangle, \ldots, C_k = \langle z_{k,1}, \ldots, z_{k,r_k}\rangle) \mapsto (G_{\mathcal{C}}, k)$
(where (i,j) indicates the j-th literal in the i-th clause),
$V(G_{\mathcal{C}}) = \{(i,j) : i \leq k, j \leq r_i\}$,
$E(G_{\mathcal{C}}) = \{(i,j),(i',j') : i = i' \text{ or } z_{i,j} = \overline{z}_{i',j'}\}$.

Proof I

Reduction from SAT:
$\mathcal{C} = (C_1 = \langle z_{1,1}, \ldots, z_{1,r_1} \rangle, \ldots, C_k = \langle z_{k,1}, \ldots, z_{k,r_k} \rangle) \mapsto (G_{\mathcal{C}}, k)$
(where (i,j) indicates the j-th literal in the i-th clause),
$V(G_{\mathcal{C}}) = \{(i,j) : i \leq k, j \leq r_i\}$,
$E(G_{\mathcal{C}}) = \{(i,j),(i',j') : i = i' \text{ or } z_{i,j} = \overline{z}_{i',j'}\}$.

It's easy to see that $G_{\mathcal{C}}$ can be determined in polynomial time, and there is an independent set of size k in $G_{\mathcal{C}}$ if and only if $\mathcal{C}$ is satisfiable.

Proof I

Reduction from SAT:
$\mathcal{C} = (C_1 = \langle z_{1,1}, \ldots, z_{1,r_1} \rangle, \ldots, C_k = \langle z_{k,1}, \ldots, z_{k,r_k} \rangle) \mapsto (G_{\mathcal{C}}, k)$
(where (i,j) indicates the j-th literal in the i-th clause),
$V(G_{\mathcal{C}}) = \{(i,j) : i \leq k, j \leq r_i\}$,
$E(G_{\mathcal{C}}) = \{(i,j), (i',j') : i = i' \text{ or } z_{i,j} = \overline{z}_{i',j'}\}$.

It's easy to see that $G_{\mathcal{C}}$ can be determined in polynomial time, and there is an independent set of size k in $G_{\mathcal{C}}$ if and only if $\mathcal{C}$ is satisfiable.

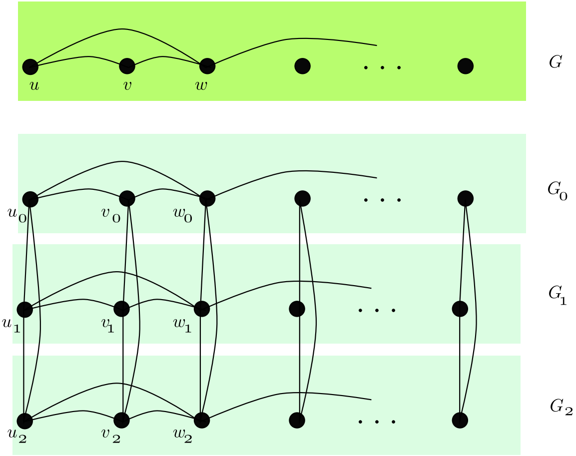
An assignment is satisfying if, for each clause, we can choose a true literal (the edges ensure that the variable and its negation do not appear together, and at most one literal is chosen from each clause).

Proof II

Proof II

Reduction from the Coloring Problem: $G \mapsto (G', |V(G)|)$, where $V(G') = \{(v,i) : v \in V(G), i \in [3]\}$ (here, (v,i) represents that vertex v is assigned color i),
$E(G') = \{(v,i)(v',i') : v = v', i \neq i' \text{ or } vv' \in E(G), i = i'\}$ (i.e., edges are used to forbid, it is forbidden that a vertex receives more than one color, or connected vertices receive the same color).

Proof II in Figure



Proof II in Words

Proof II in Words

It's easy to see that G' and $|V(G)|$ can also be determined in polynomial time, and there is an independent set of size $|V(G)|$ in G' if and only if G is 3-colorable.

Note

Note

Note

In contrast to the Coloring Problem, if k is not part of the input but a constant, then the resulting k-INDEPENDENT SET problem can be solved in polynomial time (every n-vertex graph has polynomially many k-element subsets if k is fixed).

CLIQUE, VERTEX-COVER

CLIQUE, VERTEX-COVER

Definition

CLIQUE problem: Given a graph G and a natural number k. Does G have a clique of size k?

CLIQUE, VERTEX-COVER

Definition

CLIQUE problem: Given a graph G and a natural number k. Does G have a clique of size k?

Definition

VERTEX-COVER problem: Given a graph G and a natural number k. Does G have a vertex cover of size k?

CLIQUE, VERTEX-COVER

Definition

CLIQUE problem: Given a graph G and a natural number k. Does G have a clique of size k?

Definition

VERTEX-COVER problem: Given a graph G and a natural number k. Does G have a vertex cover of size k?

Consequence

CLIQUE and VERTEX-COVER are $\mathcal{NP}$-complete.

CLIQUE, VERTEX-COVER

Definition

CLIQUE problem: Given a graph G and a natural number k. Does G have a clique of size k?

Definition

VERTEX-COVER problem: Given a graph G and a natural number k. Does G have a vertex cover of size k?

Consequence

CLIQUE and VERTEX-COVER are $\mathcal{NP}$-complete.

Equivalent to the INDEPENDENT SET problem.

HAMILTON

HAMILTON

Definition

HAMILTON, the language containing codes of graphs that have a Hamiltonian cycle.

HAMILTON

Definition

HAMILTON, the language containing codes of graphs that have a Hamiltonian cycle.

The decision problem for the language HAMILTON is to determine, given a graph, whether it has a Hamiltonian cycle.

HAMILTON

Definition

HAMILTON, the language containing codes of graphs that have a Hamiltonian cycle.

The decision problem for the language HAMILTON is to determine, given a graph, whether it has a Hamiltonian cycle.

Theorem

HAMILTON is $\mathcal{NP}$-complete.

HAMILTON

Definition

HAMILTON, the language containing codes of graphs that have a Hamiltonian cycle.

The decision problem for the language HAMILTON is to determine, given a graph, whether it has a Hamiltonian cycle.

Theorem

HAMILTON is $\mathcal{NP}$-complete.

HAMILTON is obviously in $\mathcal{NP}$.

Expanding the Statement

Expanding the Statement

We demonstrate that VERTEX-COVER can be reduced to HAMILTON.

Expanding the Statement

We demonstrate that VERTEX-COVER can be reduced to HAMILTON. That is, given a graph G and $k \in \mathbb{N}$, we can effectively define a graph R such that R has a Hamiltonian cycle if and only if G can be covered by k vertices.

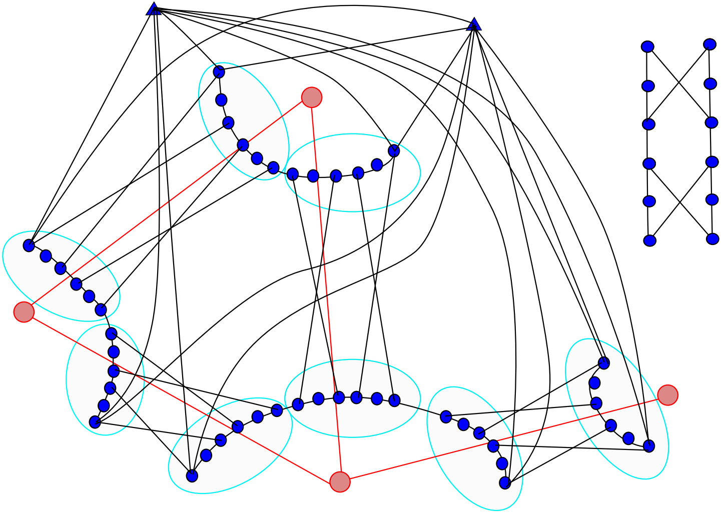
Expanding the Statement

We demonstrate that VERTEX-COVER can be reduced to HAMILTON. That is, given a graph G and $k \in \mathbb{N}$, we can effectively define a graph R such that R has a Hamiltonian cycle if and only if G can be covered by k vertices.

We illustrate the reduction with an example/figure. The general, formal description can be easily inferred from the figure, and we leave that to the interested reader.

Proof in Figure: G in red, $k = 2$, R in blue

Proof in Figure: G in red, $k=2$, R in blue



Explaining the Figure

Explaining the Figure

Each vertex in G corresponds to a path, divided into blocks of six (enclosed by light blue ellipses) containing vertices (blue circular vertices).

Explaining the Figure

Each vertex in G corresponds to a path, divided into blocks of six (enclosed by light blue ellipses) containing vertices (blue circular vertices).

Each vertex in G corresponds to a path, divided into blocks of six (enclosed by light blue ellipses) containing vertices (blue circular vertices).

Explaining the Figure

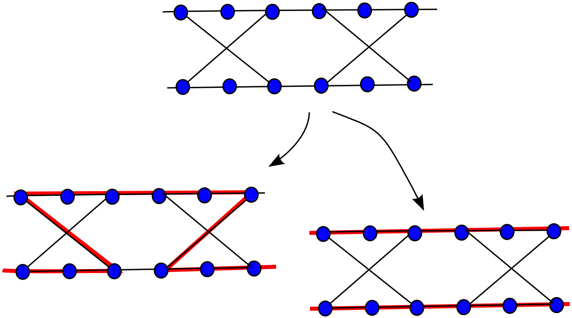
Each vertex in G corresponds to a path, divided into blocks of six (enclosed by light blue ellipses) containing vertices (blue circular vertices).

Each vertex in G corresponds to a path, divided into blocks of six (enclosed by light blue ellipses) containing vertices (blue circular vertices).

It is easy to check that for each edge, two blocks corresponding to the edge can be crossed in two different ways, as illustrated on the right.

Explaining the Figure

Explaining the Figure



(Left) Crossing occurs on a single edge. We traverse one path of a vertex, but also traverse the other block of the edge. (Right) Crossing occurs in two separate parts, each on the path of a vertex.

Proof (continued)

Proof (continued)

Assume that there exists a Hamiltonian cycle in this graph. In that case, the chosen k triangular vertices separate k vertices. The paths corresponding to these vertices traverse k blocks, with possible detours.

Proof (continued)

Assume that there exists a Hamiltonian cycle in this graph. In that case, the chosen k triangular vertices separate k vertices. The paths corresponding to these vertices traverse k blocks, with possible detours.

It's easy to verify that each edge is associated with two hexagonal blocks, and the Hamiltonian cycle can only cross these blocks in two ways.

Proof (continued)

Assume that there exists a Hamiltonian cycle in this graph. In that case, the chosen k triangular vertices separate k vertices. The paths corresponding to these vertices traverse k blocks, with possible detours.

It's easy to verify that each edge is associated with two hexagonal blocks, and the Hamiltonian cycle can only cross these blocks in two ways.

For $k = 2$, we added $k = 2$ new vertices (blue, triangular vertices).

Proof (continued)

Assume that there exists a Hamiltonian cycle in this graph. In that case, the chosen k triangular vertices separate k vertices. The paths corresponding to these vertices traverse k blocks, with possible detours.

It's easy to verify that each edge is associated with two hexagonal blocks, and the Hamiltonian cycle can only cross these blocks in two ways.

For $k = 2$, we added $k = 2$ new vertices (blue, triangular vertices).

We connect these by connecting the two end points of each path assigned to a vertex.

Proof (continued)

Proof (continued)

Now, these detours must be organized such that all other paths of vertices, as well as their blocks, are traversed.

Proof (continued)

Now, these detours must be organized such that all other paths of vertices, as well as their blocks, are traversed.

It can be easily seen that this can only be organized if the selected k vertices form a covering set.

Proof (continued)

Now, these detours must be organized such that all other paths of vertices, as well as their blocks, are traversed.

It can be easily seen that this can only be organized if the selected k vertices form a covering set.

This argument can be easily reversed.

Proof (continued)

Now, these detours must be organized such that all other paths of vertices, as well as their blocks, are traversed.

It can be easily seen that this can only be organized if the selected k vertices form a covering set.

This argument can be easily reversed.

This completes the theoretical part of the reduction. The technical details of implementation (polynomial time) are omitted.

MAX-CUT

MAX-CUT

Cutting a graph involves dividing its vertices into two disjoint parts. The cut's edge set includes edges where one endpoint is in one part, and the other endpoint is in the other part.

MAX-CUT

Cutting a graph involves dividing its vertices into two disjoint parts. The cut's edge set includes edges where one endpoint is in one part, and the other endpoint is in the other part.

Definition

MAX-CUT problem: given a graph G and a natural number k. Does G have a cut with at least k edges?

MAX-CUT

Cutting a graph involves dividing its vertices into two disjoint parts. The cut's edge set includes edges where one endpoint is in one part, and the other endpoint is in the other part.

Definition

MAX-CUT problem: given a graph G and a natural number k. Does G have a cut with at least k edges?

Theorem

MAX-CUT is $\mathcal{NP}$-complete.

MAX-CUT

Cutting a graph involves dividing its vertices into two disjoint parts. The cut's edge set includes edges where one endpoint is in one part, and the other endpoint is in the other part.

Definition

MAX-CUT problem: given a graph G and a natural number k. Does G have a cut with at least k edges?

Theorem

MAX-CUT is $\mathcal{NP}$-complete.

MAX-CUT $\in \mathcal{NP}$: a witness is a red/blue coloring, the number of edges can be calculated in polynomial time, and it can be compared with k.

Proof

Proof

MAX-CUT is $\mathcal{NP}$-hard: We reduce NOT-ALL-TRUE-3-SAT to it.

Proof

MAX-CUT is $\mathcal{NP}$-hard: We reduce NOT-ALL-TRUE-3-SAT to it.

To every 3-CNF $\mathcal{C}$, we associate the graph $G_{\mathcal{C}}$, where vertices are variables and their negations (the literals).

Proof

MAX-CUT is $\mathcal{NP}$-hard: We reduce NOT-ALL-TRUE-3-SAT to it.

To every 3-CNF $\mathcal{C}$, we associate the graph $G_{\mathcal{C}}$, where vertices are variables and their negations (the literals).

For each clause $C \in \mathcal{C}$, we connect every three literals pairwise.

Proof

MAX-CUT is $\mathcal{NP}$-hard: We reduce NOT-ALL-TRUE-3-SAT to it.

To every 3-CNF $\mathcal{C}$, we associate the graph $G_{\mathcal{C}}$, where vertices are variables and their negations (the literals).

For each clause $C \in \mathcal{C}$, we connect every three literals pairwise. (We refer to these edges as clause-edges.) Each clause corresponds to three clause-edges.

Proof

MAX-CUT is $\mathcal{NP}$-hard: We reduce NOT-ALL-TRUE-3-SAT to it.

To every 3-CNF $\mathcal{C}$, we associate the graph $G_{\mathcal{C}}$, where vertices are variables and their negations (the literals).

For each clause $C \in \mathcal{C}$, we connect every three literals pairwise. (We refer to these edges as clause-edges.) Each clause corresponds to three clause-edges. If a literal appears in multiple clauses, multiple edges will be created in the graph constructed by the reduction.

Proof

MAX-CUT is $\mathcal{NP}$-hard: We reduce NOT-ALL-TRUE-3-SAT to it.

To every 3-CNF $\mathcal{C}$, we associate the graph $G_{\mathcal{C}}$, where vertices are variables and their negations (the literals).

For each clause $C \in \mathcal{C}$, we connect every three literals pairwise. (We refer to these edges as clause-edges.) Each clause corresponds to three clause-edges. If a literal appears in multiple clauses, multiple edges will be created in the graph constructed by the reduction.

For every variable x, we draw an edge between x and $\overline{x}$. (We refer to these edges as variable-edges.) This describes all the edges of the $G_{\mathcal{C}}$ graph.

Proof (continued)

Proof (continued)

It is easy to verify that $G_{\mathcal{C}}$ can be determined in polynomial time.

Proof (continued)

It is easy to verify that $G_{\mathcal{C}}$ can be determined in polynomial time.

Furthermore, it has a cut with at least $|V| + 2|\mathcal{C}|$ edges if and only if there is an evaluation of the variables in which every clause in $\mathcal{C}$ is non-homogeneous.

Proof (continued)

It is easy to verify that $G_{\mathcal{C}}$ can be determined in polynomial time.

Furthermore, it has a cut with at least $|V| + 2|\mathcal{C}|$ edges if and only if there is an evaluation of the variables in which every clause in $\mathcal{C}$ is non-homogeneous.

Indeed, every cut of $G_{\mathcal{C}}$ has at most $|V|$ literal-edges and each clause has at most 2 out of three clause-edges. That is, $|V| + 2|\mathcal{C}|$ is an upper bound on the number of edges in any cut of $G_{\mathcal{C}}$.

Proof (continued)

It is easy to verify that $G_{\mathcal{C}}$ can be determined in polynomial time.

Furthermore, it has a cut with at least $|V| + 2|\mathcal{C}|$ edges if and only if there is an evaluation of the variables in which every clause in $\mathcal{C}$ is non-homogeneous.

Indeed, every cut of $G_{\mathcal{C}}$ has at most $|V|$ literal-edges and each clause has at most 2 out of three clause-edges. That is, $|V| + 2|\mathcal{C}|$ is an upper bound on the number of edges in any cut of $G_{\mathcal{C}}$.

If a cut (I, H) achieves this upper bound, then every literal-edge is included, meaning each variable x and $\overline{x}$ falls into either I or H. Thus, the cut defines an evaluation of our variables.

Proof (continued)

It is easy to verify that $G_{\mathcal{C}}$ can be determined in polynomial time.

Furthermore, it has a cut with at least $|V| + 2|\mathcal{C}|$ edges if and only if there is an evaluation of the variables in which every clause in $\mathcal{C}$ is non-homogeneous.

Indeed, every cut of $G_{\mathcal{C}}$ has at most $|V|$ literal-edges and each clause has at most 2 out of three clause-edges. That is, $|V| + 2|\mathcal{C}|$ is an upper bound on the number of edges in any cut of $G_{\mathcal{C}}$.

If a cut (I, H) achieves this upper bound, then every literal-edge is included, meaning each variable x and $\overline{x}$ falls into either I or H. Thus, the cut defines an evaluation of our variables.

Furthermore, every clause has two out of three clause-edges in the cut, meaning the described evaluation satisfies every clause in $\mathcal{C}$ in a non-all-true manner.

CUT Notes

CUT Notes

The MIN-CUT problem tests whether there is a cut with at most k edges. This problem can be solved in polynomial time. Using the theory of flows, the set of edges defining the smallest cut can be determined.

CUT Notes

The MIN-CUT problem tests whether there is a cut with at most k edges. This problem can be solved in polynomial time. Using the theory of flows, the set of edges defining the smallest cut can be determined.

We also note that our reduction created a graph whose optimal cut was a balanced split. This implies that the MAX-BISECTION problem is also $\mathcal{NP}$-complete. Complementing this gives us that the MAX-BISECTION and MIN-BISECTION problems are (in polynomial time) equivalent. Specifically, the MIN-BISECTION problem is also $\mathcal{NP}$-complete.

Break

Set Systems

Set Systems

Definition: Simple Set System

$\mathcal{H}$ is a simple set system over the set V if $\mathcal{H} \subseteq \mathcal{P}(V)$. The elements of $\mathcal{H}$ are the edges of the set system.

Set Systems

Definition: Simple Set System

$\mathcal{H}$ is a simple set system over the set V if $\mathcal{H} \subseteq \mathcal{P}(V)$. The elements of $\mathcal{H}$ are the edges of the set system.

A k-uniform set system is a set system where all edges have size k. Thus, simple graphs correspond precisely to 2-uniform set systems.

Set Systems

Definition: Simple Set System

$\mathcal{H}$ is a simple set system over the set V if $\mathcal{H} \subseteq \mathcal{P}(V)$. The elements of $\mathcal{H}$ are the edges of the set system.

A k-uniform set system is a set system where all edges have size k. Thus, simple graphs correspond precisely to 2-uniform set systems.

Definition: Set System

$(V, \mathcal{E}, I)$ is a set system if $I \subset V \times \mathcal{E}$ is a matching relation between the vertex set V and the edge set $\mathcal{E}$.

Set Systems

Definition: Simple Set System

$\mathcal{H}$ is a simple set system over the set V if $\mathcal{H} \subseteq \mathcal{P}(V)$. The elements of $\mathcal{H}$ are the edges of the set system.

A k-uniform set system is a set system where all edges have size k. Thus, simple graphs correspond precisely to 2-uniform set systems.

Definition: Set System

$(V, \mathcal{E}, I)$ is a set system if $I \subset V \times \mathcal{E}$ is a matching relation between the vertex set V and the edge set $\mathcal{E}$.

For every $E \in \mathcal{E}$, there is a subset $\{v \in V : v \in E\}$ that is a subset of V.

Alternative Descriptions of Set Systems

Alternative Descriptions of Set Systems

Observation

A set system $\mathcal{H}$ over V can be easily described by a bipartite graph B. The two color classes are V (upper points) and $\mathcal{H}$ (lower points). An element of the base set is connected to an element of $\mathcal{H}$ if and only if it belongs to the corresponding edge.

Alternative Descriptions of Set Systems

Observation

A set system $\mathcal{H}$ over V can be easily described by a bipartite graph B. The two color classes are V (upper points) and $\mathcal{H}$ (lower points). An element of the base set is connected to an element of $\mathcal{H}$ if and only if it belongs to the corresponding edge.

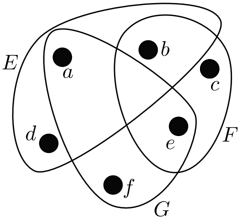
The set system can be encoded by the vertex-edge incidence matrix. This is an $n \times m$ binary matrix, where $n = |V|$ and $m = |\mathcal{H}|$.

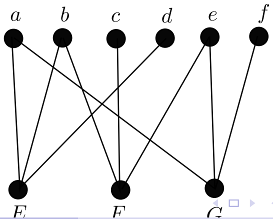
Alternative Descriptions of Set Systems

Observation

A set system $\mathcal{H}$ over V can be easily described by a bipartite graph B. The two color classes are V (upper points) and $\mathcal{H}$ (lower points). An element of the base set is connected to an element of $\mathcal{H}$ if and only if it belongs to the corresponding edge.

The set system can be encoded by the vertex-edge incidence matrix. This is an $n \times m$ binary matrix, where $n = |V|$ and $m = |\mathcal{H}|$.





$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \end{array} \begin{array}{c} \begin{array}{ccc} E & F & G \end{array} \\ \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \end{array}$$

INDEPENDENT-NODES-IN-SET-SYSTEM

INDEPENDENT-NODES-IN-SET-SYSTEM

Definition

The concept of an independent node set in graph theory can be extended to set systems in two ways:

I is independent if, for every $E \in \mathcal{H}$, $E \not\subseteq I$.

I is independent* if, for every $E \in \mathcal{H}$, $|E \cap I| \leq 1$.

INDEPENDENT-NODES-IN-SET-SYSTEM

Definition

The concept of an independent node set in graph theory can be extended to set systems in two ways:

I is independent if, for every $E \in \mathcal{H}$, $E \not\subseteq I$.

I is independent* if, for every $E \in \mathcal{H}$, $|E \cap I| \leq 1$.

Definition

INDEPENDENT-NODES-IN-SET-SYSTEM=

$\{\lceil V, \mathcal{H}, k\rceil$: there exists an independent node set I with $|I| = k\}$.

INDEPENDENT-NODES-IN-SET-SYSTEM

Definition

The concept of an independent node set in graph theory can be extended to set systems in two ways:

I is independent if, for every $E \in \mathcal{H}$, $E \not\subseteq I$.

I is independent* if, for every $E \in \mathcal{H}$, $|E \cap I| \leq 1$.

Definition

INDEPENDENT-NODES-IN-SET-SYSTEM=

$\{\lceil V, \mathcal{H}, k \rceil$: there exists an independent node set I with $|I| = k\}$.

INDEPENDENT*-NODES-IN-SET-SYSTEM=

$\{\lceil V, \mathcal{H}, k \rceil$: there exists an independent* node set I with $|I| = k\}$.

Set Systems Harder Than Graphs

Set Systems Harder Than Graphs

Theorem

(i) INDEPENDENT-NODES $\preceq$ INDEPENDENT-NODES-IN-SET-SYSTEM,

(ii) INDEPENDENT-NODES $\preceq$ INDEPENDENT*-NODES-IN-SET-SYSTEM.

Set Systems Harder Than Graphs

Theorem

(i) INDEPENDENT-NODES $\preceq$ INDEPENDENT-NODES-IN-SET-SYSTEM,

(ii) INDEPENDENT-NODES $\preceq$ INDEPENDENT*-NODES-IN-SET-SYSTEM.

Indeed, the graph-theoretical problem graph is a special case of set systems. The concept of independence in graph theory is a special case of both types of independence in set systems.

INDEPENDENT-EDGES-IN-SET-SYSTEM

INDEPENDENT-EDGES-IN-SET-SYSTEM

Definition

INDEPENDENT-EDGES-IN-SET-SYSTEM =

$$\{\lceil V, \mathcal{H}, k \rceil : \text{there exist } k \text{ edges in } \mathcal{H} \text{ that are pairwise disjoint}\}.$$

INDEPENDENT-EDGES-IN-SET-SYSTEM

Definition

INDEPENDENT-EDGES-IN-SET-SYSTEM =

$$\{\lceil V, \mathcal{H}, k \rceil : \text{there exist } k \text{ edges in } \mathcal{H} \text{ that are pairwise disjoint}\}.$$

Note: The problem INDEPENDENT-EDGES-IN-GRAPHS, alternatively MATCHING $= \{\lceil G, k \rceil : \nu(G) \geq k\}$, is easily solvable. According to Edmonds' algorithm, this problem is in $\mathcal{P}$. Hence, the case for graphs is manageable.

INDEPENDENT-EDGES-IN-SET-SYSTEM

Definition

INDEPENDENT-EDGES-IN-SET-SYSTEM =

$\{\lceil V, \mathcal{H}, k\rceil$: there exist k edges in $\mathcal{H}$ that are pairwise disjoint$\}$.

Note: The problem INDEPENDENT-EDGES-IN-GRAPHS, alternatively MATCHING $= \{\lceil G, k\rceil : \nu(G) \geq k\}$, is easily solvable. According to Edmonds' algorithm, this problem is in $\mathcal{P}$. Hence, the case for graphs is manageable.

Theorem

INDEPENDENT*-NODES-IN-SET-SYSTEM $\preceq$ INDEPENDENT-EDGES-IN-SET-SYSTEM

Set Systems: Duality

Set Systems: Duality

Based on the bipartite graph representation, it is easy to describe independent* sets. For upper points in B, there exists a set I such that V is not covered, i.e., there is a triple $a \in A, f, f' \in I \subset F$ where a is connected to both f and f'.

Set Systems: Duality

Based on the bipartite graph representation, it is easy to describe independent* sets. For upper points in B, there exists a set I such that V is not covered, i.e., there is a triple $a \in A, f, f' \in I \subset F$ where a is connected to both f and f'.

Definition

Let B be a bipartite graph describing a set system. By exchanging upper and lower roles, we obtain the dual graph B^*. Reading B^* as a set system and restoring it, we get a dual set system with $V^* = \mathcal{H}$ and $\mathcal{H}^* = V$.

Proof

Proof

From $V, \mathcal{H}, k$, we create the dual set system, keeping the value k: $V^*, \mathcal{H}^*, k$.

Proof

From $V, \mathcal{H}, k$, we create the dual set system, keeping the value k: $V^*, \mathcal{H}^*, k$.

We need to decide whether there are k upper nodes in the original set system described by the bipartite graph B, such that they do not support $\vee$ shapes.

Proof

From $V, \mathcal{H}, k$, we create the dual set system, keeping the value k: $V^*, \mathcal{H}^*, k$.

We need to decide whether there are k upper nodes in the original set system described by the bipartite graph B, such that they do not support $\vee$ shapes.

The bipartite graph $V^*, \mathcal{H}^*$ is precisely the upside-down version of B.

Proof

From $V, \mathcal{H}, k$, we create the dual set system, keeping the value k: $V^*, \mathcal{H}^*, k$.

We need to decide whether there are k upper nodes in the original set system described by the bipartite graph B, such that they do not support $\vee$ shapes.

The bipartite graph $V^*, \mathcal{H}^*$ is precisely the upside-down version of B. That is, the original decision problem is equivalent to finding k lower nodes (k edges) in the upside-down B, such that they do not support $\wedge$, i.e., they are pairwise disjoint.

Proof

From $V, \mathcal{H}, k$, we create the dual set system, keeping the value k: $V^*, \mathcal{H}^*, k$.

We need to decide whether there are k upper nodes in the original set system described by the bipartite graph B, such that they do not support $\vee$ shapes.

The bipartite graph $V^*, \mathcal{H}^*$ is precisely the upside-down version of B. That is, the original decision problem is equivalent to finding k lower nodes (k edges) in the upside-down B, such that they do not support $\wedge$, i.e., they are pairwise disjoint. In other words, we need to solve the problem INDEPENDENT-EDGES-IN-SET-SYSTEM for $V^*, \mathcal{H}^*, k$.

Proof

From $V, \mathcal{H}, k$, we create the dual set system, keeping the value k: $V^*, \mathcal{H}^*, k$.

We need to decide whether there are k upper nodes in the original set system described by the bipartite graph B, such that they do not support $\vee$ shapes.

The bipartite graph $V^*, \mathcal{H}^*$ is precisely the upside-down version of B. That is, the original decision problem is equivalent to finding k lower nodes (k edges) in the upside-down B, such that they do not support $\wedge$, i.e., they are pairwise disjoint. In other words, we need to solve the problem INDEPENDENT-EDGES-IN-SET-SYSTEM for $V^*, \mathcal{H}^*, k$.

Thus, the initial transformation is the reduction proving the theorem.

TILING

TILING

Definition

$$\text{TILING} = \{\lceil V, \mathcal{H} \rceil : \text{there exist } E_1, \ldots, E_k \text{ pairwise disjoint edges such that } \dot{\cup} E_i = V\}$$

TILING

Definition

$$\text{TILING} = \{\lceil V, \mathcal{H} \rceil : \text{there exist } E_1, \ldots, E_k \text{ pairwise disjoint edges such that } \dot{\cup} E_i = V\}$$

Theorem

INDEPENDENT-EDGES-IN-SET-SYSTEM $\preceq$ TILING.

The Reduction

The Reduction

Let $V, \mathcal{H}, k$ be the input. Let S be the maximum edge size parameter. We need to decide whether there are k pairwise disjoint edges.

The Reduction

Let $V, \mathcal{H}, k$ be the input. Let S be the maximum edge size parameter. We need to decide whether there are k pairwise disjoint edges.

The construction is done in multiple steps. First, transform $\mathcal{H}$ to make it uniform: For every edge E, introduce $S - |E|$ new vertices (different vertices for different edges).

The Reduction

Let $V, \mathcal{H}, k$ be the input. Let S be the maximum edge size parameter. We need to decide whether there are k pairwise disjoint edges.

The construction is done in multiple steps. First, transform $\mathcal{H}$ to make it uniform: For every edge E, introduce $S - |E|$ new vertices (different vertices for different edges). The problem for the modified set system is obviously equivalent to the original problem.

The Reduction

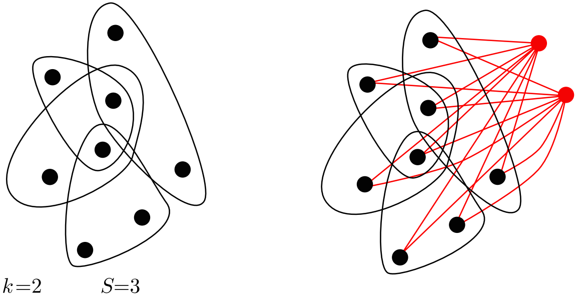
Let $V, \mathcal{H}, k$ be the input. Let S be the maximum edge size parameter. We need to decide whether there are k pairwise disjoint edges.

The construction is done in multiple steps. First, transform $\mathcal{H}$ to make it uniform: For every edge E, introduce $S - |E|$ new vertices (different vertices for different edges). The problem for the modified set system is obviously equivalent to the original problem.

In the second step, assume that $\mathcal{H}$ is a S-uniform set system. In this step, add $|V(H)| - kS$ new vertices to $V(H)$ (let $\widetilde{V}$ be the resulting set), and the elements of $\widetilde{\mathcal{H}}$ are the elements of $\mathcal{H}$ plus one set for each old-new vertex pair.

The Reduction in Pictures

The Reduction in Pictures



The second step of the reduction: $|V| - kS = 8 - 2 \cdot 3 = 2$. The two new vertices and the corresponding graph edges are shown in red on the right.

The Reduction in Words

The Reduction in Words

Observation

To tile $(\widetilde{V}, \widetilde{\mathcal{H}})$, we need to cover the $|V| - kS$ new vertices, which can only be done with $|V| - kS$ new vertex pairs. The remaining tiling edges can only be old edges, covering kS vertices. Thus, the tiling gives k independent edges in $\mathcal{H}$.

The Reduction in Words

Observation

To tile $(\widetilde{V}, \widetilde{\mathcal{H}})$, we need to cover the $|V| - kS$ new vertices, which can only be done with $|V| - kS$ new vertex pairs. The remaining tiling edges can only be old edges, covering kS vertices. Thus, the tiling gives k independent edges in $\mathcal{H}$.

Inverting the reasoning of the observation completes the theoretical part of the proof.

Matchings: The Case of 3-Uniform Set Systems

Matchings: The Case of 3-Uniform Set Systems

Definition: 3-UNIFORM-SET-SYSTEM-PARTITION

Given a 3-uniform set system. Is there a subset that is a partition of the base set?

Matchings: The Case of 3-Uniform Set Systems

Definition: 3-UNIFORM-SET-SYSTEM-PARTITION

Given a 3-uniform set system. Is there a subset that is a partition of the base set?

Definition: PERFECT-TRIPLE

Given three sets A, B, C of size k each and their transversals forming a 3-uniform set system ($\mathcal{H} \subset A \times B \times C$). Is there a set of k pairwise disjoint triples in the set system?

Matchings: The Case of 3-Uniform Set Systems

Definition: 3-UNIFORM-SET-SYSTEM-PARTITION

Given a 3-uniform set system. Is there a subset that is a partition of the base set?

Definition: PERFECT-TRIPLE

Given three sets A, B, C of size k each and their transversals forming a 3-uniform set system ($\mathcal{H} \subset A \times B \times C$). Is there a set of k pairwise disjoint triples in the set system?

Theorem

3-UNIFORM-SET-SYSTEM-PARTITION and PERFECT-TRIPLE are both $\mathcal{N}P$-complete.

3-SAT $\preceq$ PERFECT-TRIPLE $\preceq$ 3-UNIFORM-SET-SYSTEM-PARTITION

3-SAT $\preceq$ PERFECT-TRIPLE $\preceq$ 3-UNIFORM-SET-SYSTEM-PARTITION

It is sufficient to prove the first reduction.

3-SAT $\preceq$ PERFECT-TRIPLE $\preceq$ 3-UNIFORM-SET-SYSTEM-PARTITION

It is sufficient to prove the first reduction.

For this, take a φ 3-SAT input.

3-SAT $\preceq$ PERFECT-TRIPLE $\preceq$ 3-UNIFORM-SET-SYSTEM-PARTITION

It is sufficient to prove the first reduction.

For this, take a φ 3-SAT input. Assume that the variable x occurs n times.

3-SAT $\preceq$ PERFECT-TRIPLE $\preceq$ 3-UNIFORM-SET-SYSTEM-PARTITION

It is sufficient to prove the first reduction.

For this, take a φ 3-SAT input. Assume that the variable x occurs n times. From $A \cup B$, select $2n$ vertices for x:
$a_0, b_0, a_1, b_1, \ldots, a_{n-1}, b_{n-1}$.

3-SAT $\preceq$ PERFECT-TRIPLE $\preceq$ 3-UNIFORM-SET-SYSTEM-PARTITION

It is sufficient to prove the first reduction.

For this, take a φ 3-SAT input. Assume that the variable x occurs n times. From $A \cup B$, select $2n$ vertices for x: $a_0, b_0, a_1, b_1, \dots, a_{n-1}, b_{n-1}$. Consider

$$X^+ = \{\{a_0, b_0, c_0^x\}, \dots, \{a_{n-1}, b_{n-1}, c_{n-1}^x\}\},$$

$$\overline{X} = \{\{a_0, b_0, c_0^{\overline{x}}\}, \dots, \{a_{n-1}, b_{n-1}, c_{n-1}^{\overline{x}}\}\},$$

3-SAT $\preceq$ PERFECT-TRIPLE $\preceq$ 3-UNIFORM-SET-SYSTEM-PARTITION

It is sufficient to prove the first reduction.

For this, take a φ 3-SAT input. Assume that the variable x occurs n times. From $A \cup B$, select $2n$ vertices for x: $a_0, b_0, a_1, b_1, \dots, a_{n-1}, b_{n-1}$. Consider

$$X^+ = \{\{a_0, b_0, c_0^x\}, \dots, \{a_{n-1}, b_{n-1}, c_{n-1}^x\}\},$$

$$\overline{X} = \{\{a_0, b_0, c_0^{\overline{x}}\}, \dots, \{a_{n-1}, b_{n-1}, c_{n-1}^{\overline{x}}\}\},$$

Promise: *The a_i, b_i vertices will only be in the triples.*

3-SAT $\preceq$ PERFECT-TRIPLE $\preceq$ 3-UNIFORM-SET-SYSTEM-PARTITION

It is sufficient to prove the first reduction.

For this, take a φ 3-SAT input. Assume that the variable x occurs n times. From $A \cup B$, select $2n$ vertices for x: $a_0, b_0, a_1, b_1, \ldots, a_{n-1}, b_{n-1}$. Consider

$$X^+ = \{\{a_0, b_0, c_0^x\}, \ldots, \{a_{n-1}, b_{n-1}, c_{n-1}^x\}\},$$

$$\overline{X} = \{\{a_0, b_0, c_0^{\overline{x}}\}, \ldots, \{a_{n-1}, b_{n-1}, c_{n-1}^{\overline{x}}\}\},$$

Promise: *The a_i, b_i vertices will only be in the triples.*
Thus, if there is a perfect triple, either X or $\overline{X}$ is part of the triple.

3-SAT $\preceq$ PERFECT-TRIPLE $\preceq$ 3-UNIFORM-SET-SYSTEM-PARTITION

It is sufficient to prove the first reduction.

For this, take a φ 3-SAT input. Assume that the variable x occurs n times. From $A \cup B$, select $2n$ vertices for x: $a_0, b_0, a_1, b_1, \dots, a_{n-1}, b_{n-1}$. Consider

$$X^+ = \{\{a_0, b_0, c_0^x\}, \dots, \{a_{n-1}, b_{n-1}, c_{n-1}^x\}\},$$

$$\overline{X} = \{\{a_0, b_0, c_0^{\overline{x}}\}, \dots, \{a_{n-1}, b_{n-1}, c_{n-1}^{\overline{x}}\}\},$$

Promise: *The a_i, b_i vertices will only be in the triples.*
Thus, if there is a perfect triple, either X or $\overline{X}$ is part of the triple. Specifically, either all c_i^x or all $c_i^{\overline{x}}$ remain unmatched.

3-SAT $\preceq$ PERFECT-TRIPLE $\preceq$ 3-UNIFORM-SET-SYSTEM-PARTITION

It is sufficient to prove the first reduction.

For this, take a φ 3-SAT input. Assume that the variable x occurs n times. From $A \cup B$, select $2n$ vertices for x: $a_0, b_0, a_1, b_1, \ldots, a_{n-1}, b_{n-1}$. Consider

$$X^+ = \{\{a_0, b_0, c_0^x\}, \ldots, \{a_{n-1}, b_{n-1}, c_{n-1}^x\}\},$$

$$\overline{X} = \{\{a_0, b_0, c_0^{\overline{x}}\}, \ldots, \{a_{n-1}, b_{n-1}, c_{n-1}^{\overline{x}}\}\},$$

Promise: *The a_i, b_i vertices will only be in the triples.*
Thus, if there is a perfect triple, either X or $\overline{X}$ is part of the triple. Specifically, either all c_i^x or all $c_i^{\overline{x}}$ remain unmatched. The unmatched c_i indices can be considered as the value of the evaluated literal.

3-SAT $\preceq$ PERFECT-TRIPLE $\preceq$ 3-UNIFORM-SET-SYSTEM-PARTITION

It is sufficient to prove the first reduction.

For this, take a φ 3-SAT input. Assume that the variable x occurs n times. From $A \cup B$, select $2n$ vertices for x: $a_0, b_0, a_1, b_1, \ldots, a_{n-1}, b_{n-1}$. Consider

$$X^+ = \{\{a_0, b_0, c_0^x\}, \ldots, \{a_{n-1}, b_{n-1}, c_{n-1}^x\}\},$$

$$\overline{X} = \{\{a_0, b_0, c_0^{\overline{x}}\}, \ldots, \{a_{n-1}, b_{n-1}, c_{n-1}^{\overline{x}}\}\},$$

Promise: *The a_i, b_i vertices will only be in the triples.* Thus, if there is a perfect triple, either X or $\overline{X}$ is part of the triple. Specifically, either all c_i^x or all $c_i^{\overline{x}}$ remain unmatched. The unmatched c_i indices can be considered as the value of the evaluated literal. Define sets for each variable with disjoint vertex sets.

3-SAT $\preceq$ PERFECT-TRIPLE $\preceq$ 3-UNIFORM-SET-SYSTEM-PARTITION

It is sufficient to prove the first reduction.

For this, take a φ 3-SAT input. Assume that the variable x occurs n times. From $A \cup B$, select $2n$ vertices for x: $a_0, b_0, a_1, b_1, \ldots, a_{n-1}, b_{n-1}$. Consider

$$X^+ = \{\{a_0, b_0, c_0^x\}, \ldots, \{a_{n-1}, b_{n-1}, c_{n-1}^x\}\},$$

$$\overline{X} = \{\{a_0, b_0, c_0^{\overline{x}}\}, \ldots, \{a_{n-1}, b_{n-1}, c_{n-1}^{\overline{x}}\}\},$$

Promise: *The a_i, b_i vertices will only be in the triples.*
Thus, if there is a perfect triple, either X or $\overline{X}$ is part of the triple. Specifically, either all c_i^x or all $c_i^{\overline{x}}$ remain unmatched. The unmatched c_i indices can be considered as the value of the evaluated literal. Define sets for each variable with disjoint vertex sets. If there is a triple, it describes an assignment for the variables.

Proof of 3-SAT $\preceq$ PERFECT-TRIPLE (continued)

Proof of 3-SAT $\preceq$ PERFECT-TRIPLE (continued)

Let $C = \langle z_1, z_2, z_3 \rangle$ be the φ, 3-SAT input clause.

Proof of 3-SAT $\preceq$ PERFECT-TRIPLE (continued)

Let $C = \langle z_1, z_2, z_3 \rangle$ be the φ, 3-SAT input clause.

Introduce a a_C, b_C vertex and choose a c_i vertex, not chosen yet for another clause, with the upper index z_1.

Proof of 3-SAT $\preceq$ PERFECT-TRIPLE (continued)

Let $C = \langle z_1, z_2, z_3 \rangle$ be the φ, 3-SAT input clause.

Introduce a a_C, b_C vertex and choose a c_i vertex, not chosen yet for another clause, with the upper index z_1. Do the same for the other two literals of C.

Proof of 3-SAT $\preceq$ PERFECT-TRIPLE (continued)

Let $C = \langle z_1, z_2, z_3 \rangle$ be the φ, 3-SAT input clause.

Introduce a a_C, b_C vertex and choose a c_i vertex, not chosen yet for another clause, with the upper index z_1. Do the same for the other two literals of C.

By this, for clause C, we have introduced two new vertices and three new triples.

Proof of 3-SAT $\preceq$ PERFECT-TRIPLE (continued)

Let $C = \langle z_1, z_2, z_3 \rangle$ be the φ, 3-SAT input clause.

Introduce a a_C, b_C vertex and choose a c_i vertex, not chosen yet for another clause, with the upper index z_1. Do the same for the other two literals of C.

By this, for clause C, we have introduced two new vertices and three new triples. Repeat this for every clause.

Proof of 3-SAT $\preceq$ PERFECT-TRIPLE (continued)

Let $C = \langle z_1, z_2, z_3 \rangle$ be the φ, 3-SAT input clause.

Introduce a a_C, b_C vertex and choose a c_i vertex, not chosen yet for another clause, with the upper index z_1. Do the same for the other two literals of C.

By this, for clause C, we have introduced two new vertices and three new triples. Repeat this for every clause.

If in the constructed triples we find a perfect triple that covers all a and b vertices, then φ is satisfiable.

Proof of 3-SAT $\preceq$ PERFECT-TRIPLE (continued)

Let $C = \langle z_1, z_2, z_3 \rangle$ be the φ, 3-SAT input clause.

Introduce a a_C, b_C vertex and choose a c_i vertex, not chosen yet for another clause, with the upper index z_1. Do the same for the other two literals of C.

By this, for clause C, we have introduced two new vertices and three new triples. Repeat this for every clause.

If in the constructed triples we find a perfect triple that covers all a and b vertices, then φ is satisfiable. The reasoning is reversible.

Proof of 3-SAT $\preceq$ PERFECT-TRIPLE (continued)

Proof of 3-SAT $\preceq$ PERFECT-TRIPLE (continued)

In the constructed triple system, the number of a vertices and b vertices is the same.

Proof of 3-SAT $\preceq$ PERFECT-TRIPLE (continued)

In the constructed triple system, the number of a vertices and b vertices is the same. If the number of c vertices is more, add new a vertices and b vertices for balancing.

Proof of 3-SAT $\preceq$ PERFECT-TRIPLE (continued)

In the constructed triple system, the number of a vertices and b vertices is the same. If the number of c vertices is more, add new a vertices and b vertices for balancing. If the number of c vertices is less, add new c vertices for balancing.

Proof of 3-SAT $\preceq$ PERFECT-TRIPLE (continued)

In the constructed triple system, the number of a vertices and b vertices is the same. If the number of c vertices is more, add new a vertices and b vertices for balancing. If the number of c vertices is less, add new c vertices for balancing.

If new a vertices and b vertices were added to our set system, complete them with triples in all possible ways with the c vertices.

Proof of 3-SAT $\preceq$ PERFECT-TRIPLE (continued)

In the constructed triple system, the number of a vertices and b vertices is the same. If the number of c vertices is more, add new a vertices and b vertices for balancing. If the number of c vertices is less, add new c vertices for balancing.

If new a vertices and b vertices were added to our set system, complete them with triples in all possible ways with the c vertices.

If new c vertices were added to our set system, the formula is unsatisfiable.

Proof of 3-SAT $\preceq$ PERFECT-TRIPLE (continued)

In the constructed triple system, the number of a vertices and b vertices is the same. If the number of c vertices is more, add new a vertices and b vertices for balancing. If the number of c vertices is less, add new c vertices for balancing.

If new a vertices and b vertices were added to our set system, complete them with triples in all possible ways with the c vertices.

If new c vertices were added to our set system, the formula is unsatisfiable. Leave the new vertices isolated.

Proof of 3-SAT $\preceq$ PERFECT-TRIPLE (continued)

In the constructed triple system, the number of a vertices and b vertices is the same. If the number of c vertices is more, add new a vertices and b vertices for balancing. If the number of c vertices is less, add new c vertices for balancing.

If new a vertices and b vertices were added to our set system, complete them with triples in all possible ways with the c vertices.

If new c vertices were added to our set system, the formula is unsatisfiable. Leave the new vertices isolated.

The resulting 3-uniform balanced triple set system is the outcome of the reduction.

Proof of 3-SAT $\preceq$ PERFECT-TRIPLE (continued)

In the constructed triple system, the number of *a* vertices and *b* vertices is the same. If the number of *c* vertices is more, add new *a* vertices and *b* vertices for balancing. If the number of *c* vertices is less, add new *c* vertices for balancing.

If new *a* vertices and *b* vertices were added to our set system, complete them with triples in all possible ways with the *c* vertices.

If new *c* vertices were added to our set system, the formula is unsatisfiable. Leave the new vertices isolated.

The resulting 3-uniform balanced triple set system is the outcome of the reduction. The reduction proves the theorem.

Proof of 3-SAT $\preceq$ PERFECT-TRIPLE (continued)

In the constructed triple system, the number of a vertices and b vertices is the same. If the number of c vertices is more, add new a vertices and b vertices for balancing. If the number of c vertices is less, add new c vertices for balancing.

If new a vertices and b vertices were added to our set system, complete them with triples in all possible ways with the c vertices.

If new c vertices were added to our set system, the formula is unsatisfiable. Leave the new vertices isolated.

The resulting 3-uniform balanced triple set system is the outcome of the reduction. The reduction proves the theorem. The necessary claims for this are easily verifiable.

SET SYSTEM COLORING

SET SYSTEM COLORING

Definition

SET SYSTEM COLORING: given a set system H and a natural number k. Can the elements of $V(H)$ be colored with k colors such that no set in H is monochromatic?

SET SYSTEM COLORING

Definition

SET SYSTEM COLORING: given a set system H and a natural number k. Can the elements of $V(H)$ be colored with k colors such that no set in H is monochromatic?

Theorem

SET SYSTEM COLORING is $\mathcal{NP}$-complete.

SET SYSTEM COLORING

Definition

SET SYSTEM COLORING: given a set system H and a natural number k. Can the elements of $V(H)$ be colored with k colors such that no set in H is monochromatic?

Theorem

SET SYSTEM COLORING is $\mathcal{NP}$-complete.

Generalization of graph coloring problem.

SET SYSTEM 2-COLORABILITY

SET SYSTEM 2-COLORABILITY

Reminder: For graphs, the case of 2-colorability was easy to handle.

SET SYSTEM 2-COLORABILITY

Reminder: For graphs, the case of 2-colorability was easy to handle.

Definition

SET SYSTEM 2-COLORABILITY: Given a set system $\mathcal{H}$. Decide: Can the elements of $V(H)$ be colored with 2 colors such that no set in H is monochromatic?

SET SYSTEM 2-COLORABILITY

Reminder: For graphs, the case of 2-colorability was easy to handle.

Definition

SET SYSTEM 2-COLORABILITY: Given a set system $\mathcal{H}$. Decide: Can the elements of $V(H)$ be colored with 2 colors such that no set in H is monochromatic?

Theorem

TILING $\preceq$ SET SYSTEM 2-COLORABILITY.

The Reduction

The Reduction

Given an input $V, \mathcal{H}$ for the tiling problem.

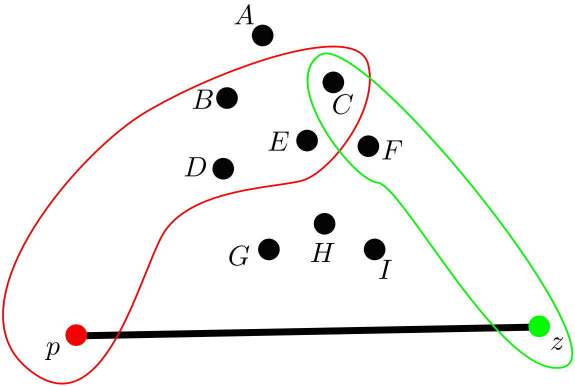
The Reduction

Given an input $V, \mathcal{H}$ for the tiling problem.

Construction: $\widetilde{V} = \mathcal{H} \cup \{p, z\}$. For $\widetilde{H}$, for every intersecting pair E, F of $\mathcal{H}$ edges, add $Z_{E,F} = \{E, F, z\}$ as an edge. For every $v \in V$, add $R_v = \{E : v \in E \in \mathcal{H}\} \cup \{p\}$ as an edge in $\widetilde{\mathcal{H}}$. Also, add the edge $\{p, z\}$.

The Reduction in Pictures

The Reduction in Pictures



$A, B, C, \ldots, H, I$ precisely represent the edges of our set system. B, C, D, E precisely represent edges containing the element a. C and F are intersecting edges. The edges inferred from the above information are drawn in the figure, which includes a fraction of the reduction.

The Reduction in Words

The Reduction in Words

Observation

In a 2-coloring of $\widetilde{V}, \widetilde{\mathcal{H}}$, let p be colored red and z be colored green (the edge $\{p, z\}$ enforces using the entire palette). The green color on the corresponding vertices of the original edges picks an edge set.

The Reduction in Words

Observation

In a 2-coloring of $\widetilde{V}, \widetilde{\mathcal{H}}$, let p be colored red and z be colored green (the edge $\{p, z\}$ enforces using the entire palette). The green color on the corresponding vertices of the original edges picks an edge set.

Indeed, two intersecting edges would lead to a green-homogeneous edge of type $Z_{E,F}$ in the reduction.

The Reduction in Words

Observation

In a 2-coloring of $\widetilde{V}, \widetilde{\mathcal{H}}$, let p be colored red and z be colored green (the edge $\{p, z\}$ enforces using the entire palette). The green color on the corresponding vertices of the original edges picks an edge set.

Indeed, two intersecting edges would lead to a green-homogeneous edge of type $Z_{E,F}$ in the reduction. While an uncovered v vertex (in the original set system) would create a red-homogeneous edge R_v.

The Reduction in Words

Observation

In a 2-coloring of $\widetilde{V}, \widetilde{\mathcal{H}}$, let p be colored red and z be colored green (the edge $\{p, z\}$ enforces using the entire palette). The green color on the corresponding vertices of the original edges picks an edge set.

Indeed, two intersecting edges would lead to a green-homogeneous edge of type $Z_{E,F}$ in the reduction. While an uncovered v vertex (in the original set system) would create a red-homogeneous edge R_v.

The reasoning is reversible, completing the proof.

Break

DIOPHANTINE INEQUALITY SYSTEM

DIOPHANTINE INEQUALITY SYSTEM

Definition: DIOPHANTINE INEQUALITY SYSTEM

Given an integer-coefficient linear inequality system $Ax \leq b$. Does it have an integer solution?

DIOPHANTINE INEQUALITY SYSTEM

Definition: DIOPHANTINE INEQUALITY SYSTEM

Given an integer-coefficient linear inequality system $Ax \leq b$. Does it have an integer solution?

Theorem

DIOPHANTINE INEQUALITY SYSTEM is $\mathcal{NP}$-complete.

DIOPHANTINE INEQUALITY SYSTEM

Definition: DIOPHANTINE INEQUALITY SYSTEM

Given an integer-coefficient linear inequality system $Ax \leq b$. Does it have an integer solution?

Theorem

DIOPHANTINE INEQUALITY SYSTEM is $\mathcal{NP}$-complete.

A witness for $\mathcal{NP}$-completeness is an integer solution.

SAT $\preceq$ DIOPHANTINE INEQUALITY SYSTEM

SAT $\preceq$ DIOPHANTINE INEQUALITY SYSTEM

Given a conjunctive normal form.

SAT $\preceq$ DIOPHANTINE INEQUALITY SYSTEM

Given a conjunctive normal form.

For each variable x_i, introduce the inequality $0 \leq x_i \leq 1$.

SAT $\preceq$ DIOPHANTINE INEQUALITY SYSTEM

Given a conjunctive normal form.

For each variable x_i, introduce the inequality $0 \leq x_i \leq 1$.

For each $\langle z_1, \ldots, z_k \rangle$ clause, introduce the inequality $t_1 + \ldots + t_k \geq 1$, where $t_i = x_j$ if $z_i = x_j$ and $t_i = 1 - x_j$ if $z_i = \overline{x}_j$.

SAT $\preceq$ DIOPHANTINE INEQUALITY SYSTEM

Given a conjunctive normal form.

For each variable x_i, introduce the inequality $0 \leq x_i \leq 1$.

For each $\langle z_1, \ldots, z_k \rangle$ clause, introduce the inequality $t_1 + \ldots + t_k \geq 1$, where $t_i = x_j$ if $z_i = x_j$ and $t_i = 1 - x_j$ if $z_i = \overline{x}_j$.

It is easy to see that the resulting inequality system can be constructed in polynomial time.

SAT $\preceq$ DIOPHANTINE INEQUALITY SYSTEM

Given a conjunctive normal form.

For each variable x_i, introduce the inequality $0 \leq x_i \leq 1$.

For each $\langle z_1, \ldots, z_k \rangle$ clause, introduce the inequality $t_1 + \ldots + t_k \geq 1$, where $t_i = x_j$ if $z_i = x_j$ and $t_i = 1 - x_j$ if $z_i = \overline{x}_j$.

It is easy to see that the resulting inequality system can be constructed in polynomial time.

It is also easy to see that the resulting inequality system has an integer solution if and only if the conjunctive normal form is satisfiable.

SUBSET SUM

SUBSET SUM

Definition

SUBSET SUM=

$$\{\lceil A, b\rceil : A \subset \mathbb{N}, b \in \mathbb{N}, \text{ there exists a subset } R \subset A,$$
$$\text{such that the sum of numbers in } R \text{ is } b\}.$$

SUBSET SUM

Definition

SUBSET SUM=

$$\{\lceil A, b\rceil : A \subset \mathbb{N}, b \in \mathbb{N}, \text{ there exists a subset } R \subset A, \text{ such that the sum of numbers in } R \text{ is } b\}.$$

A simple interpretation of the problem:

SUBSET SUM

Definition

SUBSET SUM=

$$\{\lceil A, b\rceil : A \subset \mathbb{N}, b \in \mathbb{N}, \text{ there exists a subset } R \subset A,$$
$$\text{such that the sum of numbers in } R \text{ is } b\}.$$

A simple interpretation of the problem: A represents the values of coins in our wallet. The code of A, b is accepted if and only if we can pay exactly b from our wallet.

SUBSET SUM

Definition

SUBSET SUM=

$$\{\lceil A, b \rceil : A \subset \mathbb{N}, b \in \mathbb{N}, \text{ there exists a subset } R \subset A, \text{ such that the sum of numbers in } R \text{ is } b\}.$$

A simple interpretation of the problem: A represents the values of coins in our wallet. The code of A, b is accepted if and only if we can pay exactly b from our wallet.

Theorem

SUBSET SUM is $\mathcal{NP}$-complete.

SUBSET SUM

Definition

SUBSET SUM=

$$\{\lceil A, b \rceil : A \subset \mathbb{N}, b \in \mathbb{N}, \text{ there exists a subset } R \subset A, \text{ such that the sum of numbers in } R \text{ is } b\}.$$

A simple interpretation of the problem: A represents the values of coins in our wallet. The code of A, b is accepted if and only if we can pay exactly b from our wallet.

Theorem

SUBSET SUM is $\mathcal{NP}$-complete.

The $\mathcal{NP}$-completeness of the problem is obvious.

TILING $\preceq$ SUBSET SUM

TILING $\preceq$ SUBSET SUM

Let $V, \mathcal{H}$ be the input for the TILING problem.

TILING $\preceq$ SUBSET SUM

Let $V, \mathcal{H}$ be the input for the TILING problem. Can we select a set of disjoint tiles/edges that can be used to cover V?

TILING $\preceq$ SUBSET SUM

Let $V, \mathcal{H}$ be the input for the TILING problem. Can we select a set of disjoint tiles/edges that can be used to cover V?

Construction: Let $w : V \to \{1, a, a^2, \ldots, a^{|V|-1}\}$ be an arbitrary bijection. Consider the value set as the place values in the a-based number system.

TILING $\preceq$ SUBSET SUM

Let $V, \mathcal{H}$ be the input for the TILING problem. Can we select a set of disjoint tiles/edges that can be used to cover V?

Construction: Let $w : V \to \{1, a, a^2, \dots, a^{|V|-1}\}$ be an arbitrary bijection. Consider the value set as the place values in the a-based number system.

For $E \in \mathcal{H}$, let $a_E = \sum_{v:v\in E} w(v)$. Let $A = \{a_E : E \in \mathcal{H}\}$ and $b = 11\dots1_a = \sum_{v:v\in V} w(v)$. This describes an input for the subset sum problem.

Proof

Proof

Observation

If we choose a to be $|\mathcal{H}| + 1$, then the numbers $a_i \in A$ are such that the carry-less calculation of any subset sum in the a-based number system can be computed.

Proof

Observation

If we choose a to be $|\mathcal{H}| + 1$, then the numbers $a_i \in A$ are such that the carry-less calculation of any subset sum in the a-based number system can be computed.

The observation immediately gives that finding a subset sum of all 1's is equivalent to the original TILING problem on $\mathcal{H}$ (for a sufficiently large a).

Proof

Observation

If we choose a to be $|\mathcal{H}|+1$, then the numbers $a_i \in A$ are such that the carry-less calculation of any subset sum in the a-based number system can be computed.

The observation immediately gives that finding a subset sum of all 1's is equivalent to the original TILING problem on $\mathcal{H}$ (for a sufficiently large a).

The largest number in the reduction is $S = \sum_{i=0}^{|V|-1} a^i = \frac{a^{|V|}-1}{a-1} < a^{|V|}$.

Proof

Observation

If we choose a to be $|\mathcal{H}| + 1$, then the numbers $a_i \in A$ are such that the carry-less calculation of any subset sum in the a-based number system can be computed.

The observation immediately gives that finding a subset sum of all 1's is equivalent to the original TILING problem on $\mathcal{H}$ (for a sufficiently large a).

The largest number in the reduction is $S = \sum_{i=0}^{|V|-1} a^i = \frac{a^{|V|}-1}{a-1} < a^{|V|}$. Its code length is $|V| \log a = |V| \log(|\mathcal{H}| + 1)$.

Proof

Observation

If we choose a to be $|\mathcal{H}|+1$, then the numbers $a_i \in A$ are such that the carry-less calculation of any subset sum in the a-based number system can be computed.

The observation immediately gives that finding a subset sum of all 1's is equivalent to the original TILING problem on $\mathcal{H}$ (for a sufficiently large a).

The largest number in the reduction is $S=\sum_{i=0}^{|V|-1} a^i = \frac{a^{|V|}-1}{a-1} < a^{|V|}$. Its code length is $|V|\log a = |V|\log(|\mathcal{H}|+1)$. Our reduction is polynomial.

KNAPSACK

KNAPSACK

Definition

KNAPSACK: Given a set of items T. Each $t \in T$ has a volume V_t and a value v_t ($v_t, V_t \in \mathbb{N}$). Given a knapsack, which can hold at most H total volume of items. Also given a value limit L. ($H, L \in \mathbb{N}$.) Can we select a subset of T to fit in the knapsack and have a total value at least L?

KNAPSACK

Definition

KNAPSACK: Given a set of items T. Each $t \in T$ has a volume V_t and a value v_t ($v_t, V_t \in \mathbb{N}$). Given a knapsack, which can hold at most H total volume of items. Also given a value limit L. ($H, L \in \mathbb{N}$.) Can we select a subset of T to fit in the knapsack and have a total value at least L?

The problem's interpretation: The set A describes the volumes of items. Given b boxes, each of which can hold at most c total volume. Can we pack the items into the boxes?

KNAPSACK

Definition

KNAPSACK: Given a set of items T. Each $t \in T$ has a volume V_t and a value v_t ($v_t, V_t \in \mathbb{N}$). Given a knapsack, which can hold at most H total volume of items. Also given a value limit L. ($H, L \in \mathbb{N}$.) Can we select a subset of T to fit in the knapsack and have a total value at least L?

The problem's interpretation: The set A describes the volumes of items. Given b boxes, each of which can hold at most c total volume. Can we pack the items into the boxes?

Theorem

KNAPSACK is $\mathcal{N}P$-complete.

KNAPSACK

Definition

KNAPSACK: Given a set of items T. Each $t \in T$ has a volume V_t and a value v_t ($v_t, V_t \in \mathbb{N}$). Given a knapsack, which can hold at most H total volume of items. Also given a value limit L. ($H, L \in \mathbb{N}$.) Can we select a subset of T to fit in the knapsack and have a total value at least L?

The problem's interpretation: The set A describes the volumes of items. Given b boxes, each of which can hold at most c total volume. Can we pack the items into the boxes?

Theorem

KNAPSACK is $\mathcal{N}P$-complete.

The $\mathcal{N}P$-completeness of the problem is obvious.

SUBSET SUM $\preceq$ KNAPSACK

SUBSET SUM $\preceq$ KNAPSACK

Given $A \subset \mathbb{N}$ as an input for the SUBSET SUM problem.

SUBSET SUM $\preceq$ KNAPSACK

Given $A \subset \mathbb{N}$ as an input for the SUBSET SUM problem.

For each $a \in A$, take an item with volume and value both equal to a.

SUBSET SUM $\preceq$ KNAPSACK

Given $A \subset \mathbb{N}$ as an input for the SUBSET SUM problem.

For each $a \in A$, take an item with volume and value both equal to a.

The volume of our knapsack is b.

SUBSET SUM $\preceq$ KNAPSACK

Given $A \subset \mathbb{N}$ as an input for the SUBSET SUM problem.

For each $a \in A$, take an item with volume and value both equal to a.

The volume of our knapsack is b.

The value limit is also b.

SUBSET SUM $\preceq$ KNAPSACK

Given $A \subset \mathbb{N}$ as an input for the SUBSET SUM problem.

For each $a \in A$, take an item with volume and value both equal to a.

The volume of our knapsack is b.

The value limit is also b.

Clearly, a subset of items can fit in the knapsack and achieve the value limit if and only if the sum is exactly b.

PARTITION

PARTITION

Definition

PARTITION: Given integers in set A. Can we split our numbers into two equal-sum subsets?

PARTITION

Definition

PARTITION: Given integers in set A. Can we split our numbers into two equal-sum subsets? Can we partition A?

PARTITION

Definition

PARTITION: Given integers in set A. Can we split our numbers into two equal-sum subsets? Can we partition A?

Theorem

PARTITION is $\mathcal{N}P$-complete.

PARTITION

Definition

PARTITION: Given integers in set A. Can we split our numbers into two equal-sum subsets? Can we partition A?

Theorem

PARTITION is $\mathcal{N}P$-complete.

The $\mathcal{N}P$-completeness of the problem is obvious.

KNAPSACK $\preceq$ PARTITION

KNAPSACK $\preceq$ PARTITION

Given $A \subset \mathbb{N}$ as an input for the KNAPSACK problem.

KNAPSACK $\preceq$ PARTITION

Given $A \subset \mathbb{N}$ as an input for the KNAPSACK problem.

Let S be the sum of numbers in A.

KNAPSACK $\preceq$ PARTITION

Given $A \subset \mathbb{N}$ as an input for the KNAPSACK problem.

Let S be the sum of numbers in A.

Examine the KNAPSACK problem on A, $b = S/2$, and $c = S/2$.

KNAPSACK $\preceq$ PARTITION

Given $A \subset \mathbb{N}$ as an input for the KNAPSACK problem.

Let S be the sum of numbers in A.

Examine the KNAPSACK problem on A, $b = S/2$, and $c = S/2$.

The knapsack can be filled with items such that both subsets have the same total value if and only if A can be partitioned into two equal-sum subsets.

This is the end!

Thank you for your attention!