جامعة
الملك سعود
King Saud University

Theory of Computation
CSC 339 – Spring 2021

Chapter-7: part4
NP-Completeness

King Saud University

Department of Computer Science

Dr. Azzam Alsudais

Introduction

- **Recall that problems in NP are decision problems that can be <u>solved in polynomial time on nondeterministic TMs</u>.**

Introduction

- **Recall that problems in NP are decision problems that can be <u>solved in polynomial time on nondeterministic TMs</u>.**

- **A polynomial time solution to some problems in NP can be used to solve all problems in NP.**

  - **These problems are called NP-complete.**

Introduction

- **Recall that problems in NP are decision problems that can be solved in polynomial time on nondeterministic TMs.**

- **On the other hand, problems in P are decision problems that can be solved in polynomial time on deterministic TMs.**

Introduction

- **Late 1960's and early 1970's, the Cook-Levin theorem was introduced:**

Introduction

- **Late 1960's and early 1970's, the Cook-Levin theorem was introduced:**
  - **NP-completeness: the complexity of some problems in NP is related to that of the entire class.**

Introduction

- **Late 1960's and early 1970's, the Cook-Levin theorem was introduced:**
  - **NP-completeness: the complexity of some problems in NP is related to that of the entire class.**
  - **SAT: The Boolean Satisfiability problem is NP-complete.**

Introduction

- **Late 1960's and early 1970's, the Cook-Levin theorem was introduced:**
  - **NP-completeness: the complexity of some problems in NP is related to that of the entire class.**
  - **SAT: The Boolean Satisfiability problem is NP-complete.**
- **What does it mean for a problem L to be NP-complete?**

Introduction

- **Late 1960's and early 1970's, the Cook-Levin theorem was introduced:**
  - **NP-completeness: the complexity of some problems in NP is related to that of the entire class.**
  - **SAT: The Boolean Satisfiability problem is NP-complete.**
- **What does it mean for a problem *L* to be NP-complete?**
  - **If a polynomial time algorithm solves an NP-complete problem, then all other problems in NP can be solved in polynomial time.**

NP-Completeness

- **What does it mean for a problem L to be NP-complete?**

NP-Completeness

- What does it mean for a problem L to be NP-complete?
  - L is in NP, and

NP-Completeness

- **What does it mean for a problem L to be NP-complete?**
  - **L is in NP, and**
  - **All problems in NP are reducible to L in polynomial time.**

NP-Completeness

- **What does it mean for a problem L to be NP-complete?**
  - **L is in NP, and**
  - **All problems in NP are reducible to L in polynomial time.**
- **Why is NP-complete important?**

NP-Completeness

- **What does it mean for a problem *L* to be NP-complete?**
  - ***L* is in NP, and**
  - **All problems in NP are reducible to *L* in polynomial time.**
- **Why is NP-complete important?**
  - **To show that P = NP, all we need to do is find a polynomial time algorithm to an NP-complete problem.**

NP-Completeness

- **What does it mean for a problem L to be NP-complete?**
  - **L is in NP, and**
  - **All problems in NP are reducible to L in polynomial time.**
- **Why is NP-complete important?**
  - **To show that P = NP, all we need to do is find a polynomial time algorithm to an NP-complete problem.**
  - **To show that P ≠ NP, if a problem in NP requires more than polynomial time, an NP-complete one does.**

NP-Completeness: Satisfiability Problem (SAT)

- **A boolean formula is an expression involving boolean variables and operations.**

NP-Completeness: Satisfiability Problem (SAT)

- **A boolean formula is an expression involving boolean <u>variables</u> and <u>operations</u>.**

- **A boolean formula is satisfiable if some assignment of 0’s and 1’s makes the formula evaluate to 1.**

NP-Completeness: Satisfiability Problem (SAT)

- **A boolean formula is an expression involving boolean <u>variables</u> and <u>operations</u>.**

- **A boolean formula is satisfiable if some assignment of 0's and 1's makes the formula evaluate to 1.**

- **Take the following formula**
  - $\varphi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$
  - **To satisfy φ, we may assign these values**
    - x = 0
    - y = 1
    - z = 0

NP-Completeness: Satisfiability Problem (SAT)

- **A boolean formula is an expression involving boolean <u>variables</u> and <u>operations</u>.**

- **A boolean formula is satisfiable if some assignment of 0's and 1's makes the formula evaluate to 1.**

- **Take the following formula**
  - $\varphi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$
  - **To satisfy φ, we may assign these values**
    - x = 0
    - y = 1
    - z = 0

Theorem 7.27

SAT $\in$ P iff P = NP.

NP-Completeness: Polynomial Time Reducibility

Definition 7.28

A function $f\colon \Sigma^* \to \Sigma^*$ is a polynomial time computable function if some polynomial time Turing machine M exists that halts with just $f(w)$ on its tape, when started on any input w.

NP-Completeness: Polynomial Time Reducibility

Definition 7.28

A function $f\colon \Sigma^* \rightarrow \Sigma^*$ is a polynomial time computable function if some polynomial time Turing machine M exists that halts with just $f(w)$ on its tape, when started on any input w.

Definition 7.29

Language A is ***polynomial time reducible***, to language B, written $A \leq_P B$, if a polynomial time computable function $f\colon \Sigma^* \rightarrow \Sigma^*$ exists, where for every w,

$$w \in A \Longleftrightarrow f(w) \in B.$$

The function f is called the ***polynomial time reduction*** of A to B.

NP-Completeness: Polynomial Time Reducibility

Definition 7.28

A function $f: \Sigma^* \rightarrow \Sigma^*$ is a polynomial time computable function if some polynomial time Turing machine M exists that halts with just $f(w)$ on its tape, when started on any input w.

The main idea is to use one problem to solve another!

written $A \leq_P B$, if a polynomial time computable function $f: \Sigma^* \rightarrow \Sigma^*$ exists, where for every w,

$$w \in A \Longleftrightarrow f(w) \in B.$$

The function f is called the ***polynomial time reduction*** of A to B.

NP-Completeness: Polynomial Time Reducibility

Theorem 7.31

If $A \leq_P B$ and $B \in P$, then $A \in P$.

NP-Completeness: Polynomial Time Reducibility

- **Let M be a polynomial time algorithm deciding B and f be the polynomial time reduction from A to B. We describe a polynomial time algorithm N deciding A as follows.**

NP-Completeness: Polynomial Time Reducibility

- **Let *M* be a polynomial time algorithm deciding *B* and *f* be the polynomial time reduction from *A* to *B*. We describe a polynomial time algorithm *N* deciding *A* as follows.**

N = "On input w:

1. Compute f (w).
2. Run M on input f (w) and output whatever M outputs."

NP-Completeness: Polynomial Time Reducibility

- **Let M be a polynomial time algorithm deciding B and f be the polynomial time reduction from A to B. We describe a polynomial time algorithm N deciding A as follows.**

N = "On input w:
1. Compute f (w).
2. Run M on input f (w) and output whatever M outputs."

- **We have $w \in A$ whenever $f(w) \in B$ because f is a reduction from A to B.**

- **M accepts $f(w)$ whenever $w \in A$.**

NP-Completeness: 3SAT

- *3SAT* **is** *SAT* **where all formulas are in a special format.**

NP-Completeness: 3SAT

- *3SAT* **is** *SAT* **where all formulas are in a special format.**
  - **A literal is a boolean variable (**x **or** $\neg x$**)**

NP-Completeness: 3SAT

- *3SAT* **is** *SAT* **where all formulas are in a special format.**
  - **A literal is a boolean variable (*x* or ¬*x*)**
  - **A clause is several literals connected with ∨s.**

NP-Completeness: 3SAT

- *3SAT* **is** *SAT* **where all formulas are in a special format.**
  - **A literal is a boolean variable (**x **or** $\neg x$**)**
  - **A clause is several literals connected with** $\vee$**s.**
  - **A Boolean formula in conjunctive normal form (called a** *cnf-formula***) connects multiple clauses with** $\wedge$**s.**

NP-Completeness: 3SAT

- *3SAT* **is** *SAT* **where all formulas are in a special format.**
  - **A literal is a boolean variable (*x* or ¬*x*)**
  - **A clause is several literals connected with ∨s.**
  - **A Boolean formula in conjunctive normal form (called a *cnf-formula*) connects multiple clauses with ∧s.**
  - ***3cnf*-formula is when all clauses contain exactly 3 literals.**
- *3SAT = {⟨φ⟩ | φ is a satisfiable 3cnf-formula}*

$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4) \wedge (x_4 \vee x_5 \vee x_6).$$

NP-Completeness: Polynomial Time Reduction (3SAT)

Theorem 7.32

3SAT is polynomial time reducible to CLIQUE.

NP-Completeness: Polynomial Time Reduction (3SAT)

Theorem 7.32

3SAT is polynomial time reducible to CLIQUE.

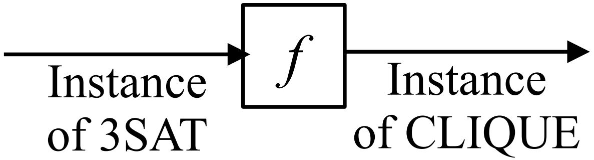
This requires converting boolean formulas to graphs

NP-Completeness: Polynomial Time Reduction (3SAT)

- **The reduction function f generates the string $\langle G, k\rangle$, where G is an undirected graph with k nodes.**

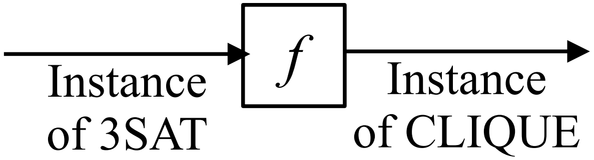
NP-Completeness: Polynomial Time Reduction (3SAT)

- **The reduction function f generates the string $\langle G, k\rangle$, where G is an undirected graph with k nodes.**



NP-Completeness: Polynomial Time Reduction (3SAT)

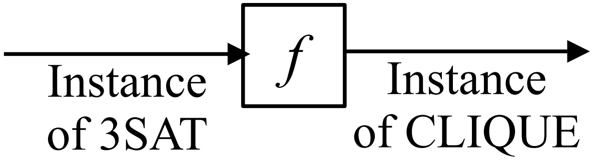
- **The reduction function f generates the string $\langle G, k \rangle$, where G is an undirected graph with k nodes.**



- **The nodes in G are organized into k groups of three nodes each called the triples, $t_1, \ldots, t_k$.**

NP-Completeness: Polynomial Time Reduction (3SAT)

- **The reduction function f generates the string $\langle G, k \rangle$, where G is an undirected graph with k nodes.**

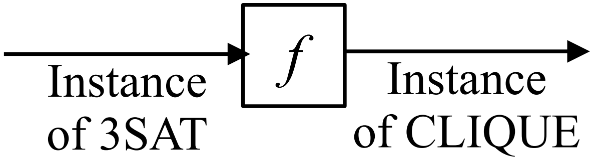


- **The nodes in G are organized into k groups of three nodes each called the triples, $t_1, \ldots, t_k$.**

- **Each triple corresponds to one of the clauses in φ.**

NP-Completeness: Polynomial Time Reduction (3SAT)

- **The reduction function** f **generates the string** $\langle G, k\rangle$**, where** G **is an undirected graph with** k **nodes.**



- **The nodes in G are organized into k groups of three nodes each called the triples,** $t_1, \ldots, t_k$**.**

- **Each triple corresponds to one of the clauses in** φ**.**

- **Each node in a triple corresponds to a literal in the associated clause.**

NP-Completeness: Polynomial Time Reduction (3SAT)

- **The edges of G connect to all but two types of pairs of nodes in G.**

NP-Completeness: Polynomial Time Reduction (3SAT)

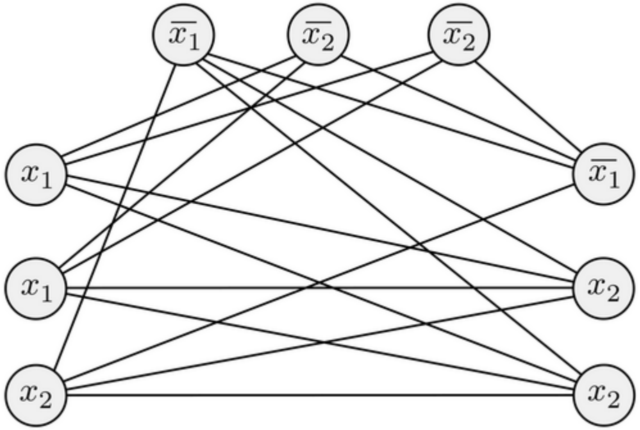
- **The edges of G connect to all but two types of pairs of nodes in G.**
  - **No edge is present between nodes in the same triple, and**

NP-Completeness: Polynomial Time Reduction (3SAT)

- **The edges of G connect to all but two types of pairs of nodes in G.**
  - **No edge is present between nodes in the same triple, and**
  - **no edge is present between two contradictory labels (e.g., x and $\neg x$).**

NP-Completeness: Polynomial Time Reduction (3SAT)

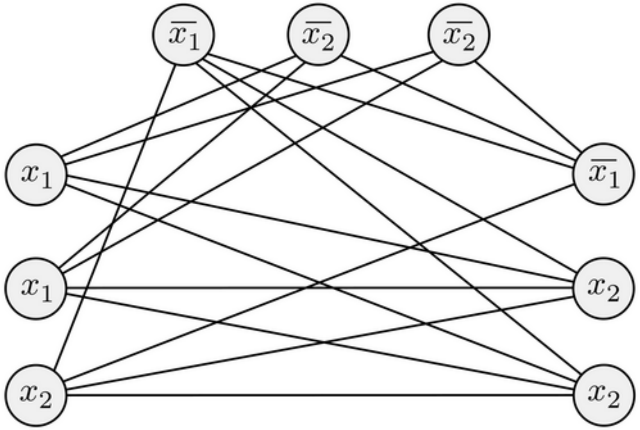
- **The edges of G connect to all but two types of pairs of nodes in G.**
  - **No edge is present between nodes in the same triple, and**
  - **no edge is present between two contradictory labels (e.g., x and $\neg x$).**
- **Example**



$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2).$$

NP-Completeness: Polynomial Time Reduction (3SAT)

- **The edges of G connect to all but two types of pairs of nodes in G.**
  - **No edge is present between nodes in the same triple, and**
  - **no edge is present between two contradictory labels (e.g., x and $\neg x$).**
- **Example**

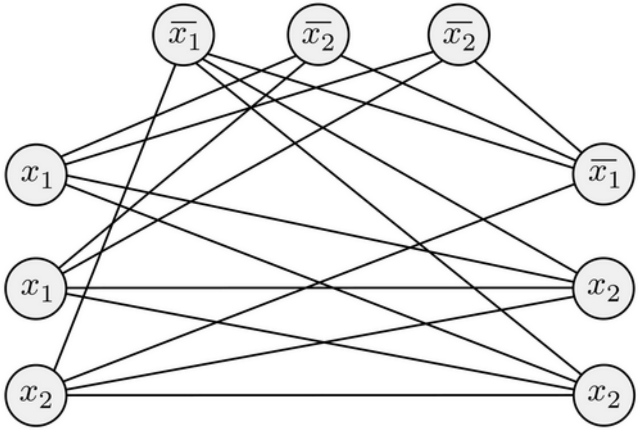


$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2).$$

φ is satisfiable iff G has a k-clique

NP-Completeness: Polynomial Time Reduction (3SAT)

- **The edges of *G* connect to all but two types of pairs of nodes in *G*.**
  - **No edge is present between nodes in the same triple, and**
  - **no edge is present between two contradictory labels (e.g., *x* and ¬*x*).**
- **Example**



φ is satisfiable iff G has a k-clique

How?

$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2).$$

NP-Completeness: Polynomial Time Reduction (3SAT)

- **Suppose that φ has a satisfying assignment.**

3SAT	CLIQUE
At least one literal must be true in every clause	Select one node corresponding to a true literal in the satisfying assignment

NP-Completeness: Polynomial Time Reduction (3SAT)

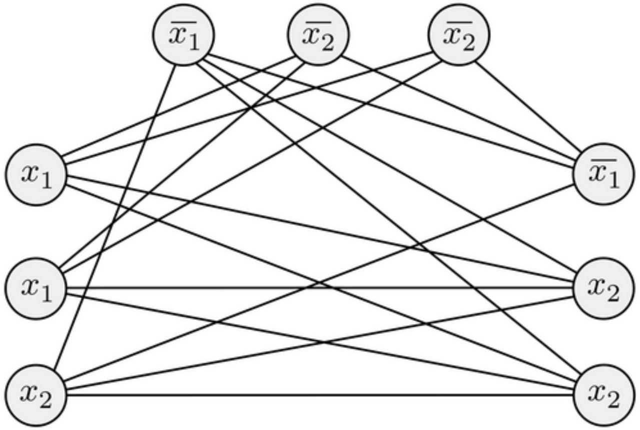
- **Suppose that φ has a satisfying assignment.**

3SAT	CLIQUE
At least one literal must be true in every clause	Select one node corresponding to a true literal in the satisfying assignment
If more than one literal is true in a particular clause, choose one arbitrarily	The nodes we select form a *k-clique*

NP-Completeness: Polynomial Time Reduction (3SAT)

- **Suppose that φ has a satisfying assignment.**

At least one literal must be true in every clause. Selecting one true literal from each clause will form a k-clique in the graph. k nodes were selected because we only chose one from each triple. Each pair is joined with an edge because it does not meet the exception given earlier. Therefore, G contains a k-clique.



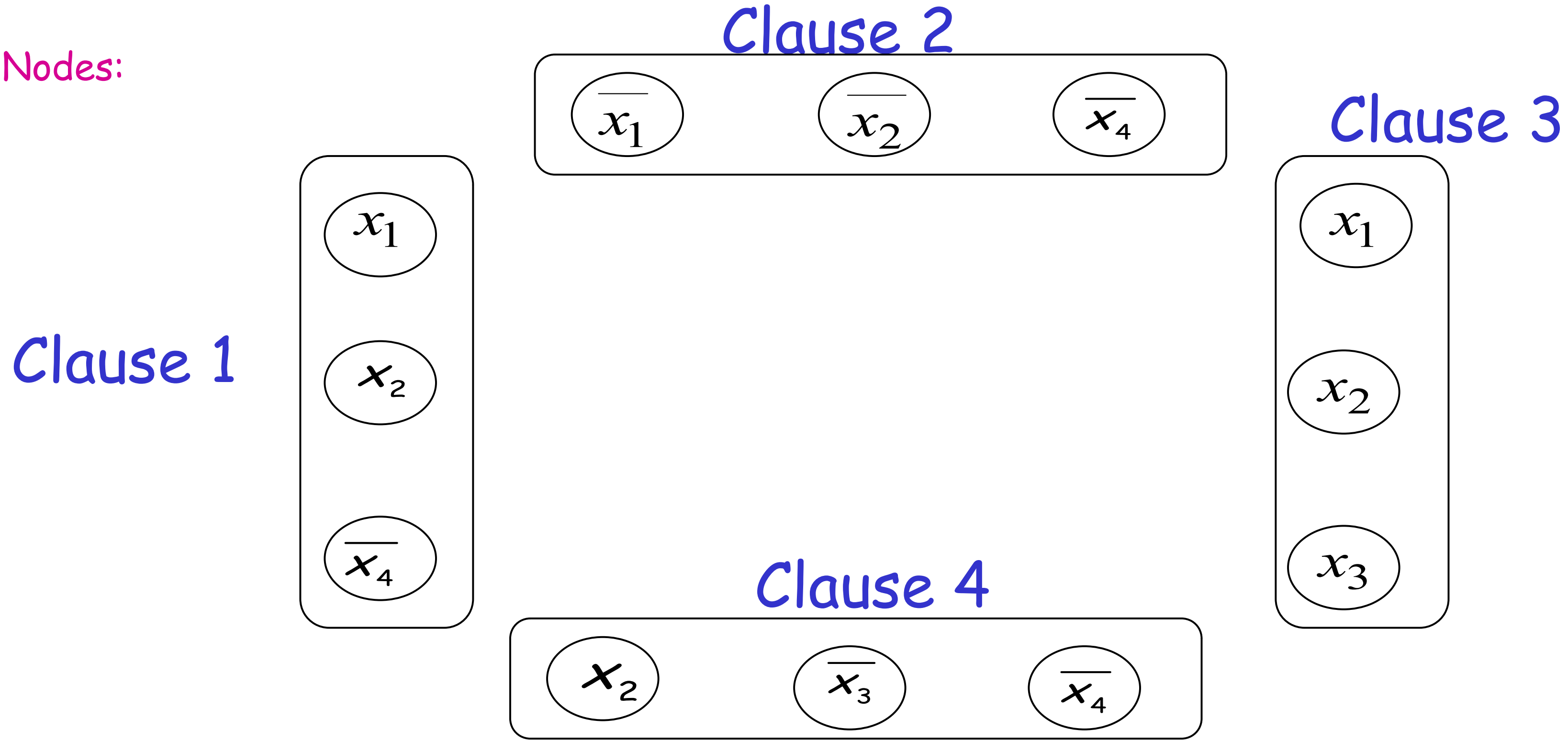
$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2).$$

Transform formula to graph.

Example:

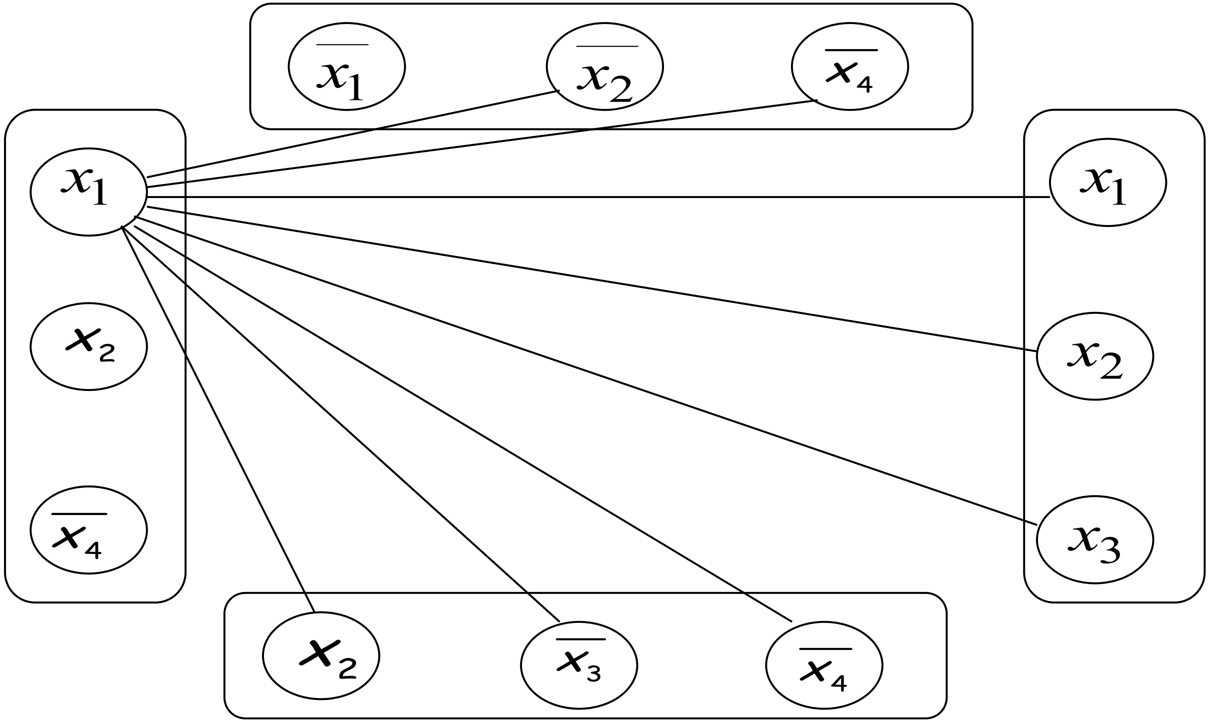
$$(x_1 \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4}) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4})$$

Create Nodes:



Slide credit to Dr. Ouiem Bchir

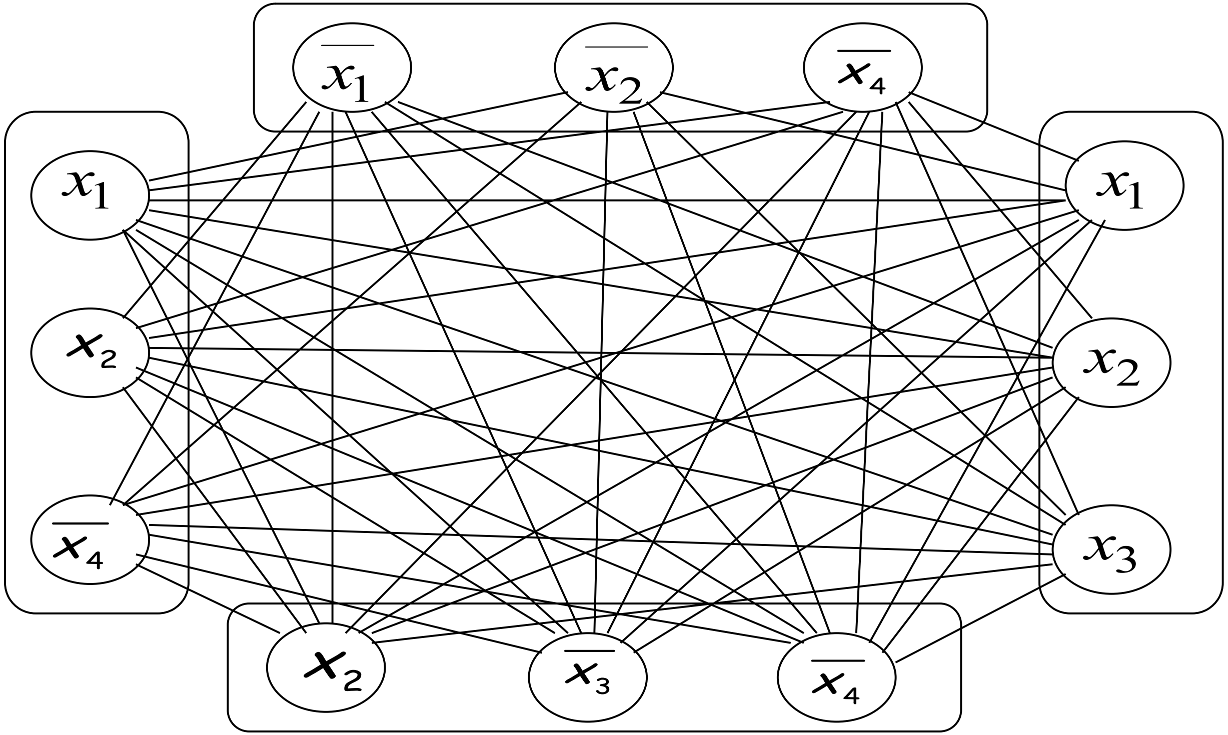
$(x_1 \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4}) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4})$



Add link from a literal ξ to a literal in every other clause, except the complement $\overline{\xi}$

Slide credit to Dr. Ouiem Bchir

$(x_1 \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4}) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4})$



Resulting Graph

Slide credit to Dr. Ouiem Bchir

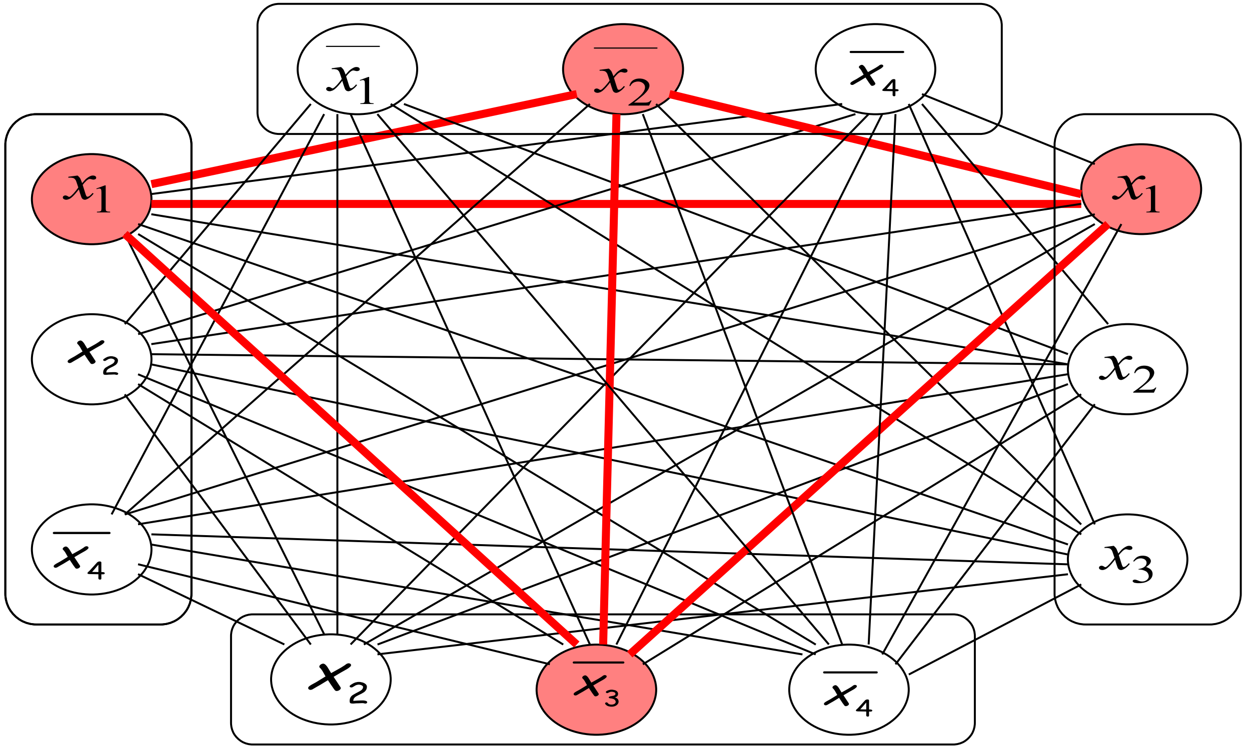
$$(x_1 \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4}) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4}) = 1$$

$x_1 = 1$

$x_2 = 0$

$x_3 = 0$

$x_4 = 1$



The formula is satisfied if and only if the Graph has a 4-clique

End of Proof

Slide credit to Dr. Ouiem Bchir

NP-Completeness: Definition

Definition 7.34

A language B is ***NP-complete*** if it satisfies two conditions:

1. B is in NP, and
2. every A in NP is polynomial time reducible to B.

NP-Completeness: Definition

Definition 7.34

A language B is ***NP-complete*** if it satisfies two conditions:

1. B is in NP, and
2. every A in NP is polynomial time reducible to B.

Theorem 7.35

If B is NP-complete and $B \in$ P, then P = NP.

NP-Completeness

Theorem 7.36

If B is NP-complete and $B \leq_{\mathrm{P}} C$ for C in NP, then C is NP-complete.

NP-Completeness

Theorem 7.36

If B is NP-complete and $B \leq_P C$ for C in NP, then C is NP-complete.

Check page 304 for proof

NP-Completeness: Cook-Levin Theorem

Cook-Levin Theorem (7.37)

SAT is NP-complete.

NP-Completeness: Cook-Levin Theorem

Cook-Levin Theorem (7.37)

SAT is NP-complete.

Proof

Part-1: Need to show that SAT is in NP.
A nondeterministic TM can find the assignment to a given formula and accept if the assignment satisfies that formula.

NP-Completeness: Cook-Levin Theorem

Cook-Levin Theorem (7.37)

SAT is NP-complete.

Proof

Part-1: Need to show that SAT is in NP.
A nondeterministic TM can find the assignment to a given formula and accept if the assignment satisfies that formula.

Part-2: Need to show that every problem in NP is polynomial reducible to SAT.