Algorithm and Data Structure Analysis (ADSA)

P and NP

Formal setting

- Inputs are encoded in some fixed alphabet Σ.
- A decision problem is a subset $L \subseteq \Sigma^*$.
- Characteristic function χ_L of L.

$$\chi_L(x) = \begin{cases} 1 & \text{if } x \in L \\ 0 & \text{if } x \notin L \end{cases}$$

$\sum^*$: Set of all possible strings over the alphabet Σ.

So far, we defined complexity classes via Turing machines in a formal way.
Let's simplify a bit and use an equivalent notation.

Class NP

A decision problem L is in NP iff there is a predicate Q(x,y) and a polynomial p such that

1. for any $x \in \Sigma^*$, $x \in L$ iff there is a $y \in \Sigma^*$ with $|y| \leq p(|x|)$ and $Q(x, y)$, and

2. Q is computable in polynomial time

y is a witness that x belongs to L (guess such a witness y).
The predicate Q(x,y) is a function that returns true iff y is a witness that x belongs to L.
Verify y in polynomial time using Q.

Hamiltonian Cycle Problem

- Given: Undirected graph $G=(V,E)$.
- Decide whether G contains a Hamiltonian cycle. A Hamiltonian cycle is cycle that visits each node exactly once and returns to the start vertex.

Example: Class NP

The Hamiltonian Cycle Problem is in NP:

- We can guess a Hamiltonian cycle y in the input graph x.
- Given such a cycle y we can check in polynomial time whether it is a Hamiltonian cycle in x.

Class **P**

- A decision problem is polynomial solvable iff its characteristic function is polynomial-time computable.
- We use **P** to denote the class of polynomial-time-solvable decision problems.

Obviously $P \subseteq NP$

One of the major open question in Computer Science: Is P=NP?

Most people believe that P≠NP.

Reduction

A decision problem L’ is polynomial-time reducible to a decision problem L if there is a polynomial time computable function g such that for all $x \in \Sigma^*$, we have

$$x \in L' \text{ iff } g(x) \in L.$$

Intuition: L is at least as hard as L’.

To solve L’, we can use the function g and a solver for L.

NP-Completeness

- A decision problem L is NP-hard iff every problem in NP is polynomial-time reducible to it.
- A decision problem is NP-complete iff it is NP-hard and in NP.

NP-Complete Problems

Cook/Levin (1971): Boolean Satisfiability is NP-complete.

The decision variants of TSP, CLIQUE, GraphColoring are NP-Complete.

How to show NP-completeness?

To show that a decision problem L is NP-complete, we need to show:

1. L in NP.
2. L is NP-hard, i.e., there is some *other* NP-complete problem L' that can be reduced to L in polynomial time.

Transitivity of reducibility relation implies that all problems in NP can be reduced to L if L is NP-complete.

Hamiltonian Cycle Problem

- Given: Undirected graph G=(V,E).
- Decide whether G contains a Hamiltonian cycle. A Hamiltonian cycle is cycle that visits each node exactly once and returns to the start vertex.

The Hamiltonian cycle problem is NP-complete!

Traveling Salesperson Problem

- Given: Complete edge-weighted undirected graph G=(V,E) and an integer C.
- Decide whether G contains a Hamiltonian cycle of cost at most C.

Show that the Traveling Salesperson Problem is NP-complete.

- Assume that the Hamiltonian cycle problem is NP-complete.
- We want to show that the Traveling Salesperson Problem (TSP) is NP-complete

Theorem: The Traveling Salesperson Problem is NP-complete.

Proof:

1. Show that TSP is in NP.
2. Show that the Hamiltonian Cycle Problem is polynomial-time reducible to the TSP.

Claim: The TSP is in NP.

Proof:

- We guess a TSP tour of cost at most C.
- We verify the tour in polynomial time by checking whether it is a TSP tour of cost at most C.

Claim: The Hamiltonian cycle problem is polynomial-time reducible to the TSP.

Proof:

- Let $G=(V,E)$ be an input to the Hamiltonian cycle problem.
- We construct a TSP $T=(V,E')$ such that G contains a Hamiltonian cycle if and only if T contains a Hamiltonian cycle of cost at most C.

- T=(V,E') is the complete graph on n nodes consisting of all possible edges.
- We have to set the edge costs c({u,v}), u≠v and the cost bound C.
- We set

$$c(\{u,v\}) = 1 \text{ iff } \{u,v\} \in E$$

$$c(\{u,v\}) = 2 \text{ iff } \{u,v\} \notin E$$

- Cost bound C=n.

- All edges in G get a cost of 1 in T.
- A Hamiltonian cycle in G is a tour of cost n in T.
- Each tour in T has cost at least n as a tour consists of n edges.
- Each tour in T that does not use all edges of G has cost at least n+1 as it uses at least one edge of cost 2.
- G contains a Hamiltonian cycle iff T contains a tour of cost n.

Boolean Satisfiability problem

- Given: A Boolean expression in conjunctive normal form.
- Decide whether it has a satisfying assignment.

Conjunctive normal form is conjunction of clauses $C_1 \wedge C_2 \wedge \ldots \wedge C_k$

Clause is disjunction of literals $l_1 \vee l_2 \vee \ldots \vee l_h$.

Literal is variable or a negated variable.

Cook/Levin (1971): Boolean Satisfiability is NP-complete.

Clique Problem

- Given: Undirected graph G=(V,E) and an integer k.
- Decide whether the graph contains a complete subgraph (clique) on k nodes.

Clique Problem

Theorem: The Clique problem is NP-complete.

Proof:

Show that

1. The clique problem is in NP.
2. The clique problem is NP-hard.

Lemma 1: The Clique Problem is in NP.

- We can guess a witness y (clique of size k) and verify in polynomial time whether it is a clique of size k in the input graph given by x.

Clique is NP-hard

Lemma 2 (see Lemma 2.10 in Mehlhorn/Sanders):

The Boolean satisfiability problem is polynomial time reducible to the clique problem.

Proof:

- Given an input F to Boolean satisfiability (formula of k clauses), we need a polynomial transformation to turn this into a graph G.
- F should have a satisfying assignment iff G has a clique of size k.

NP-hardness Clique

Let $F = C_1 \wedge \ldots \wedge C_k$ with

$$C_i = l_{i1} \vee \ldots \vee l_{ih_i}$$

$$l_{ij} = x_{ij}^{\beta_{ij}}, \ \beta_{ij} \in \{0, 1\}$$

$$x_{ij} \text{ is a variable}$$

$$\beta_{ij} = 0 \text{ indicates a negated variable}$$

be a formula in conjunctive normal form.

Transform F into a graph G!!!

Graph G:

Node set (each variable is a node)

$$V = \{r_{ij} : 1 \leq i \leq k \text{ and } 1 \leq j \leq h_i\}$$

Edge set: Two nodes are connected if they belong to different clauses and an assignment can satisfy them simultaneously (they are not a negation of each other).

Edge set:

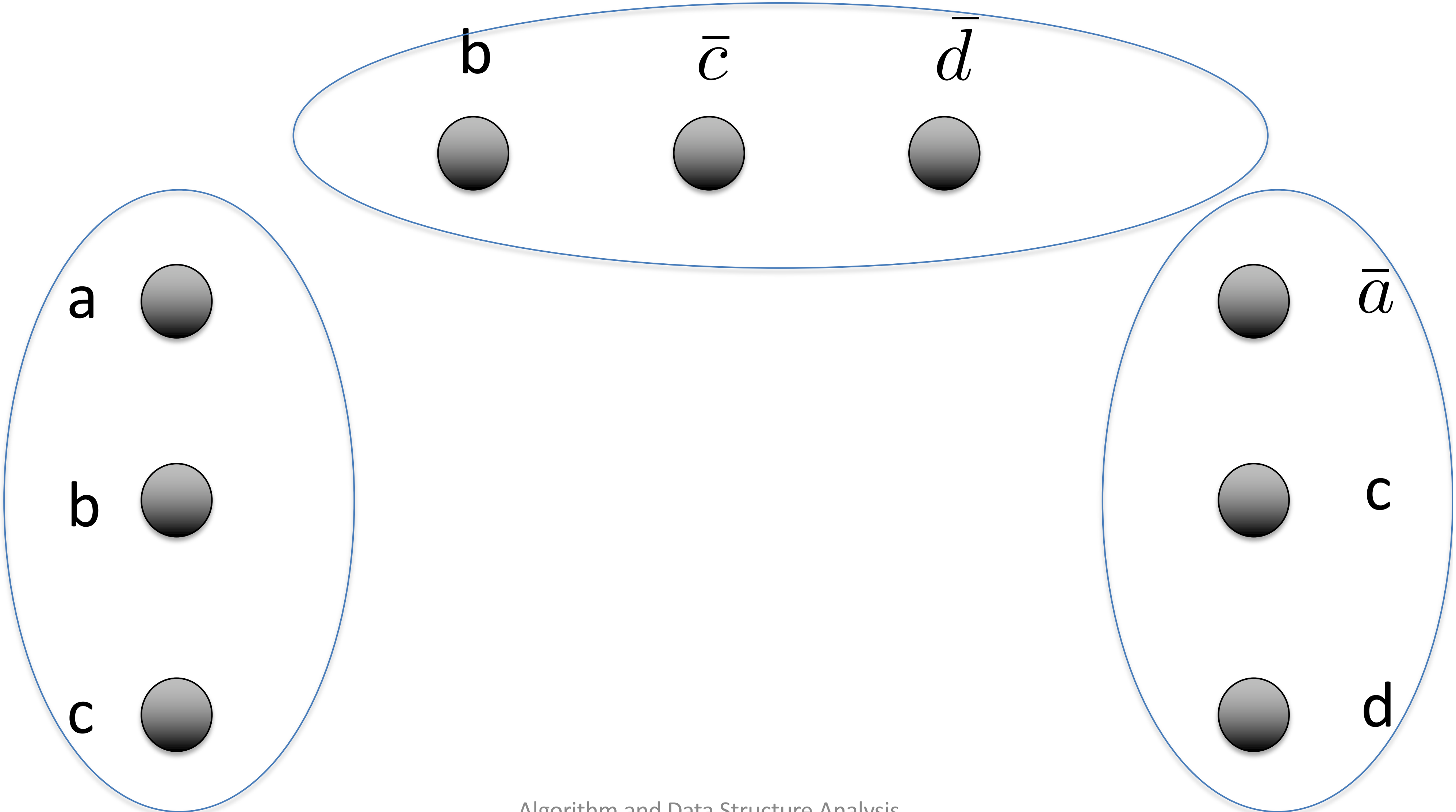
$$r_{ij} \text{ and } r_{i'j'} \text{ are connected } (\{r_{ij}, r_{i'j'}\} \in E)$$

$$\text{iff } i \neq i' \text{ and either } x_{ij} \neq x_{i'j'} \text{ or } \beta_{ij} = \beta_{i'j'}$$

Claim: F is satisfiable iff G has a clique of size k.

Example

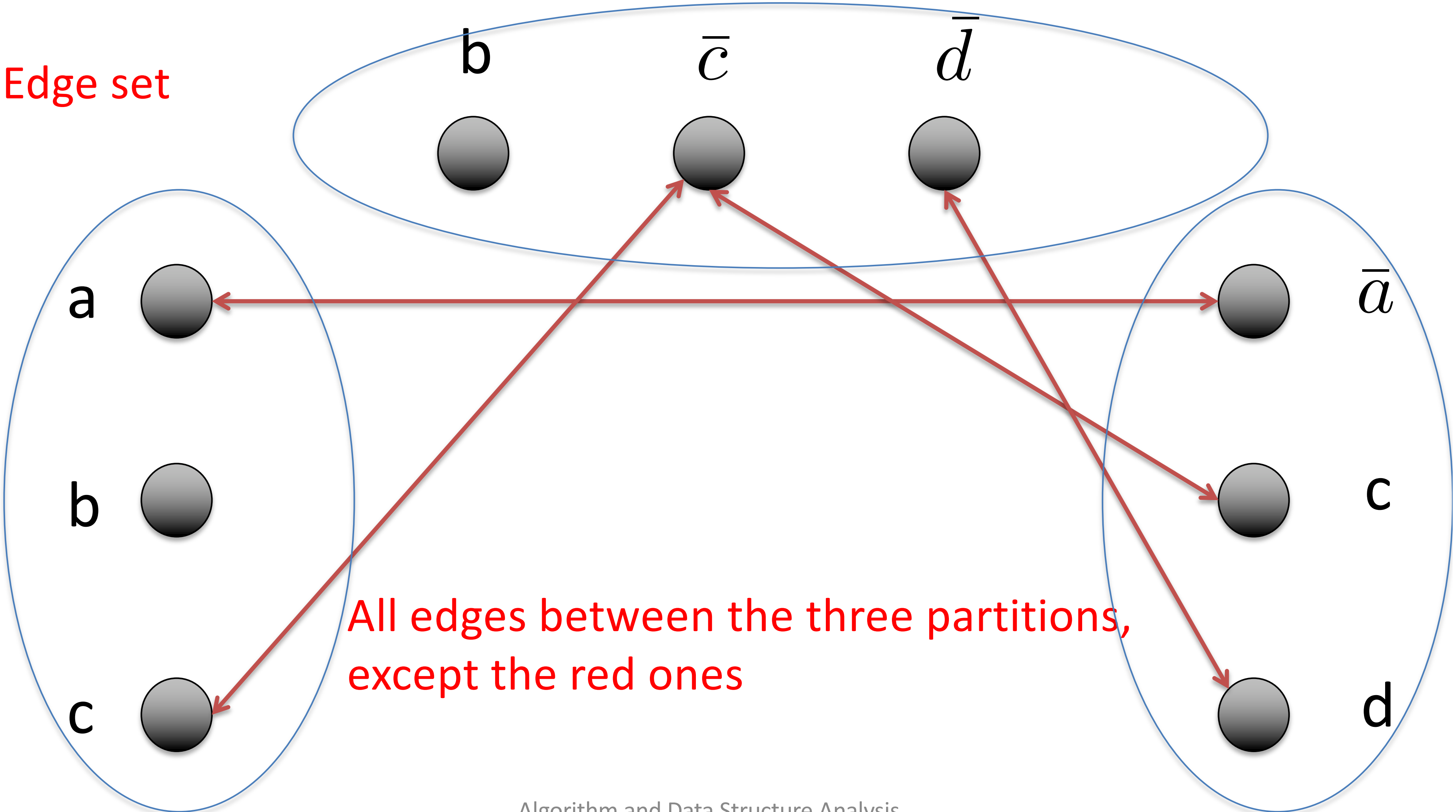
$$F = (a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d)$$



Example

$$F = (a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d)$$

Edge set



All edges between the three partitions, except the red ones

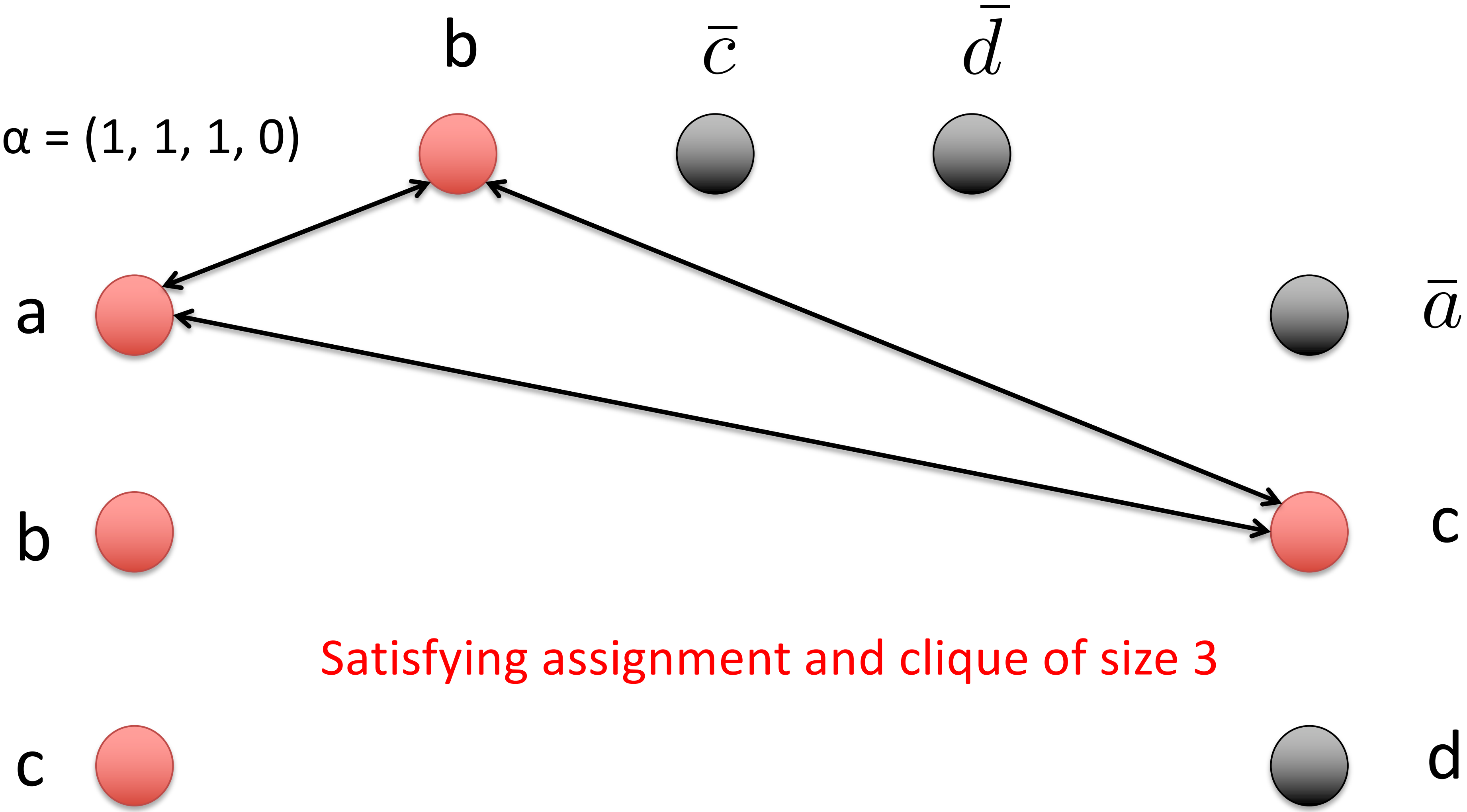
=> (Satifying assignment to clique of size k)

- Assume that there is a satisfying assignment α for F.
- The assignment must satisfy at least one literal in every clause.
- The subgraph spanned by these literals is a clique of size k.
- A missing edge would imply that two variables are in conflict and α is not a satisfying assignment (contradiction).

Example

$$F = (a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d)$$

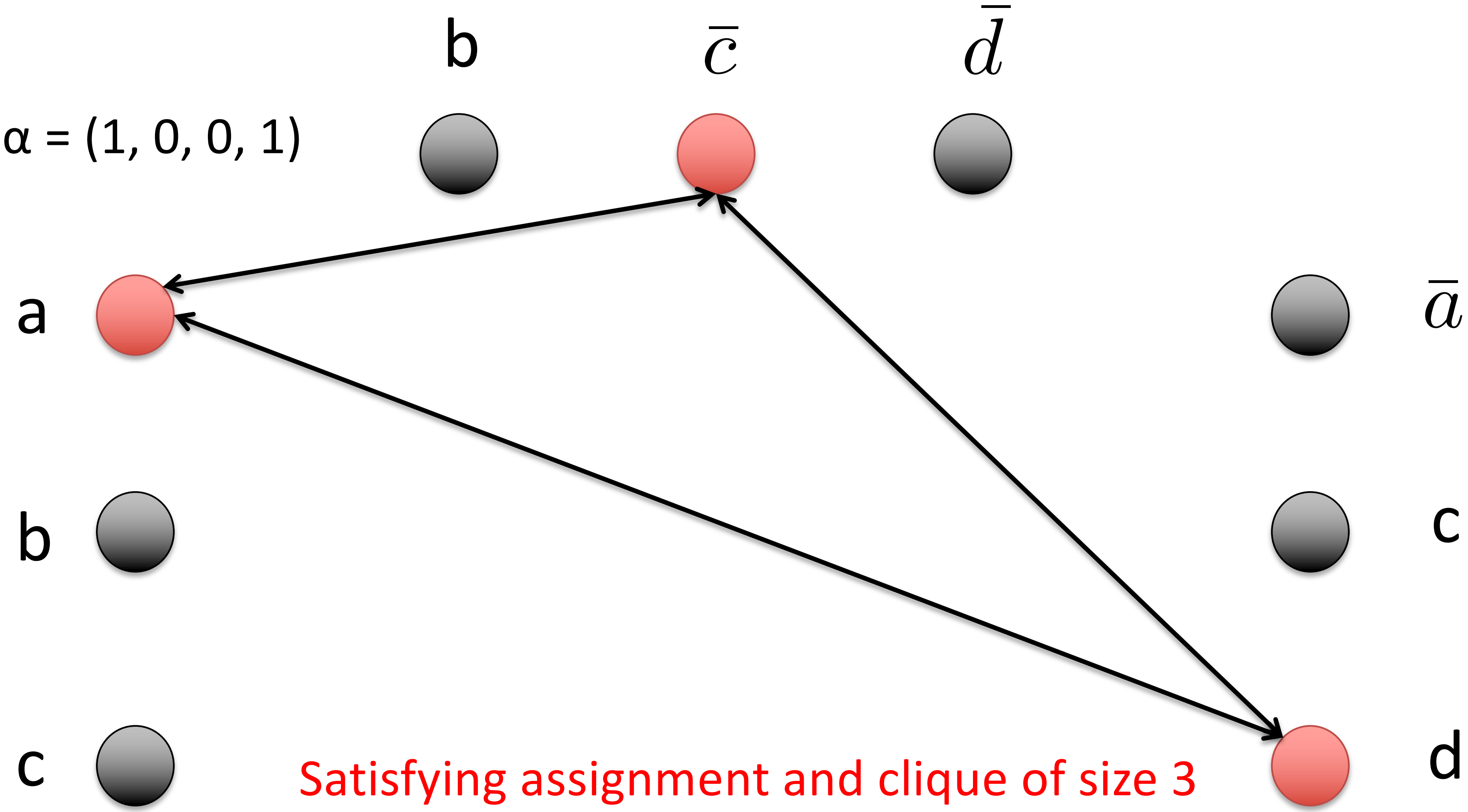
α = (1, 1, 1, 0)



Satisfying assignment and clique of size 3

Example

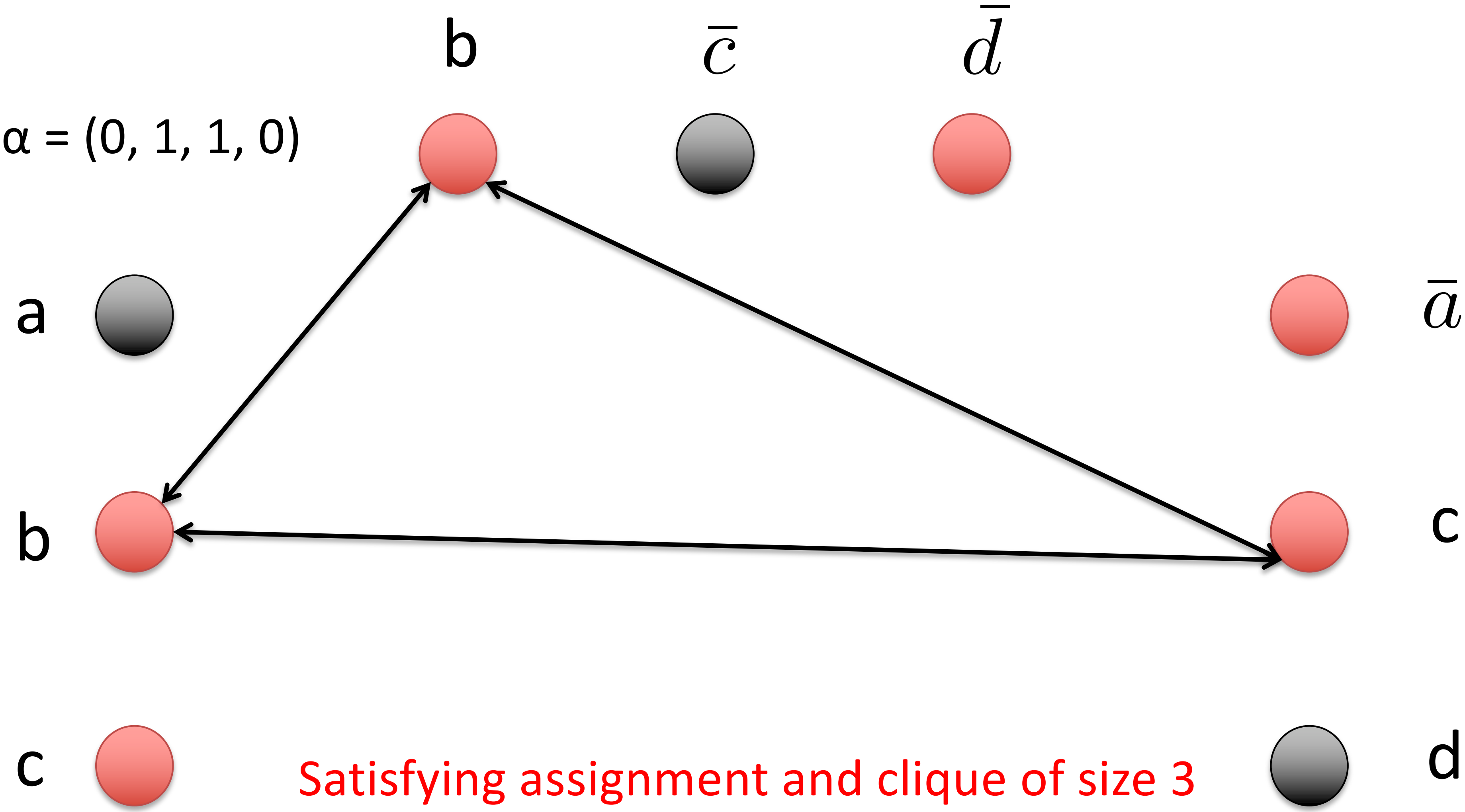
$$F = (a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d)$$



Example

$$F = (a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d)$$

α = (0, 1, 1, 0)



Satisfying assignment and clique of size 3

<= (Clique of size k to satisfying assignment)

- Assume that K is a clique of size k.
- For each clause K contains exactly one node r_{ij_i}
- We construct a satisfying assignment α by setting $\alpha(x_{ij_i}) = \beta_{ij_i}$
- α is well defined as same variable get the same value, i. e.

$$x_{ij_i} = x_{i'j_i'} \text{ implies } \beta_{ij_i} = \beta_{i'j_i'}$$

$\square$