Artificial Intelligence (CS6380)

Constraint satisfaction

Search techniques considered till now

- **BFS, DFS, Uniform cost, Greedy Best First, A*, IDA***
    - Systematic search methods
    - All are complete on finite spaces
    - Some have guarantees of optimality
- **Local search**
    - Not systematic and hence incomplete
    - Memory efficient, useful in infinite state spaces

Till now we assumed that state is atomic.

Can we have more details about the state? Use a factored representation

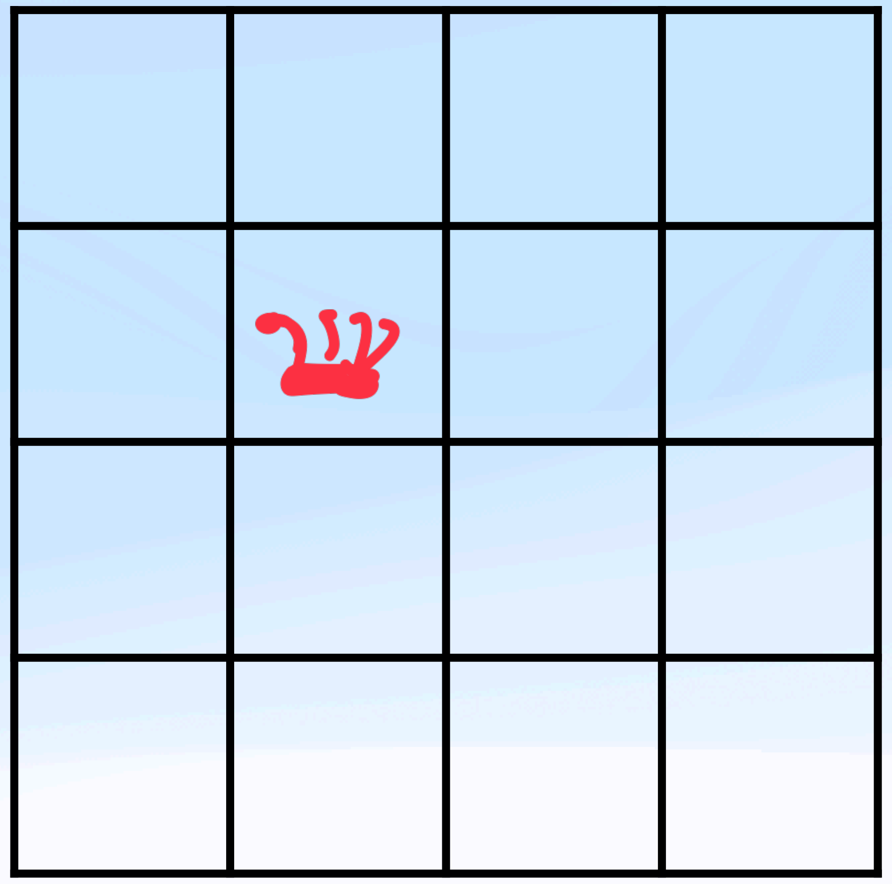
N Queens problem

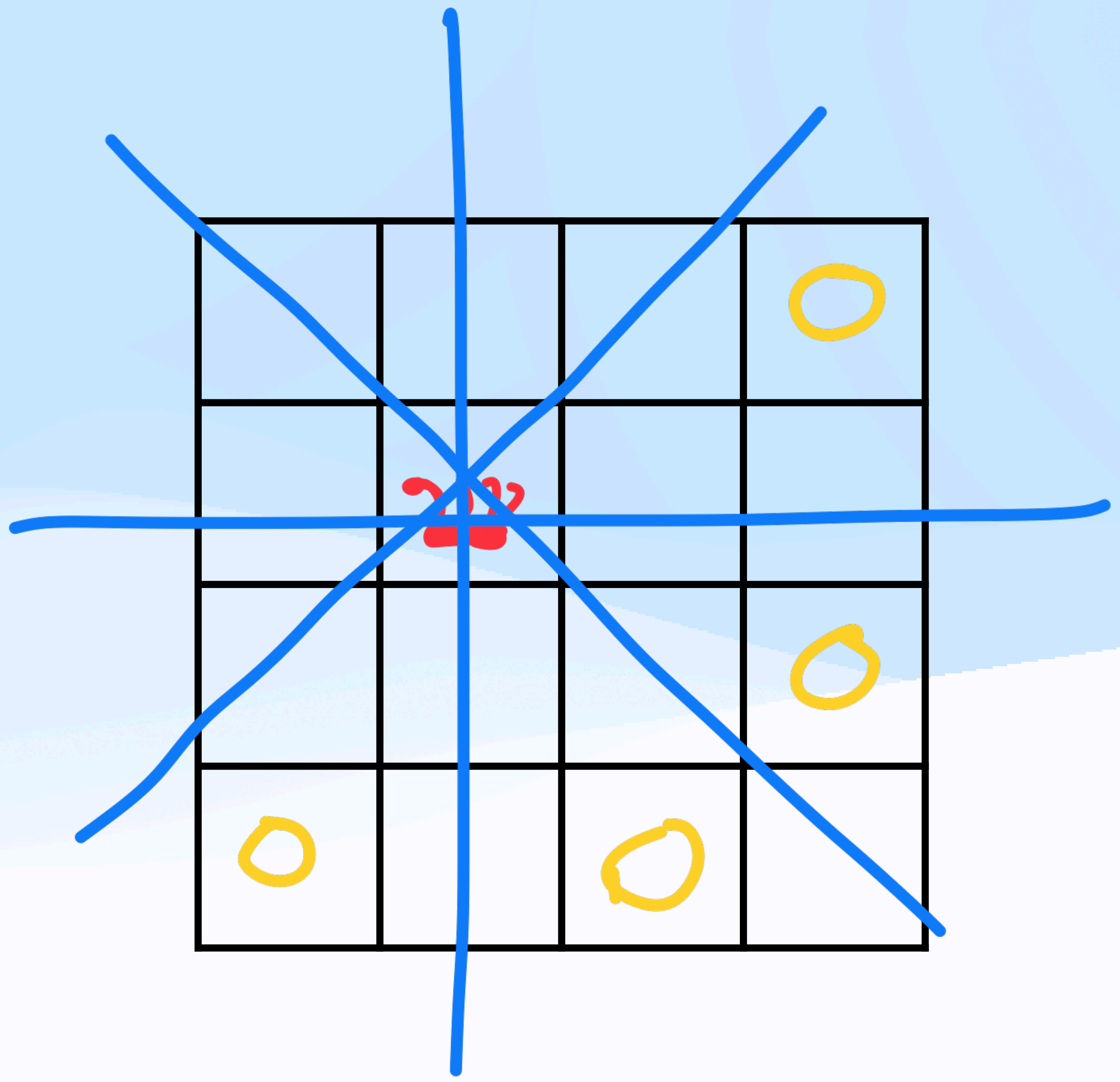
N x N empty chess board

- Place N queens such that no queen is under attack by any other queen

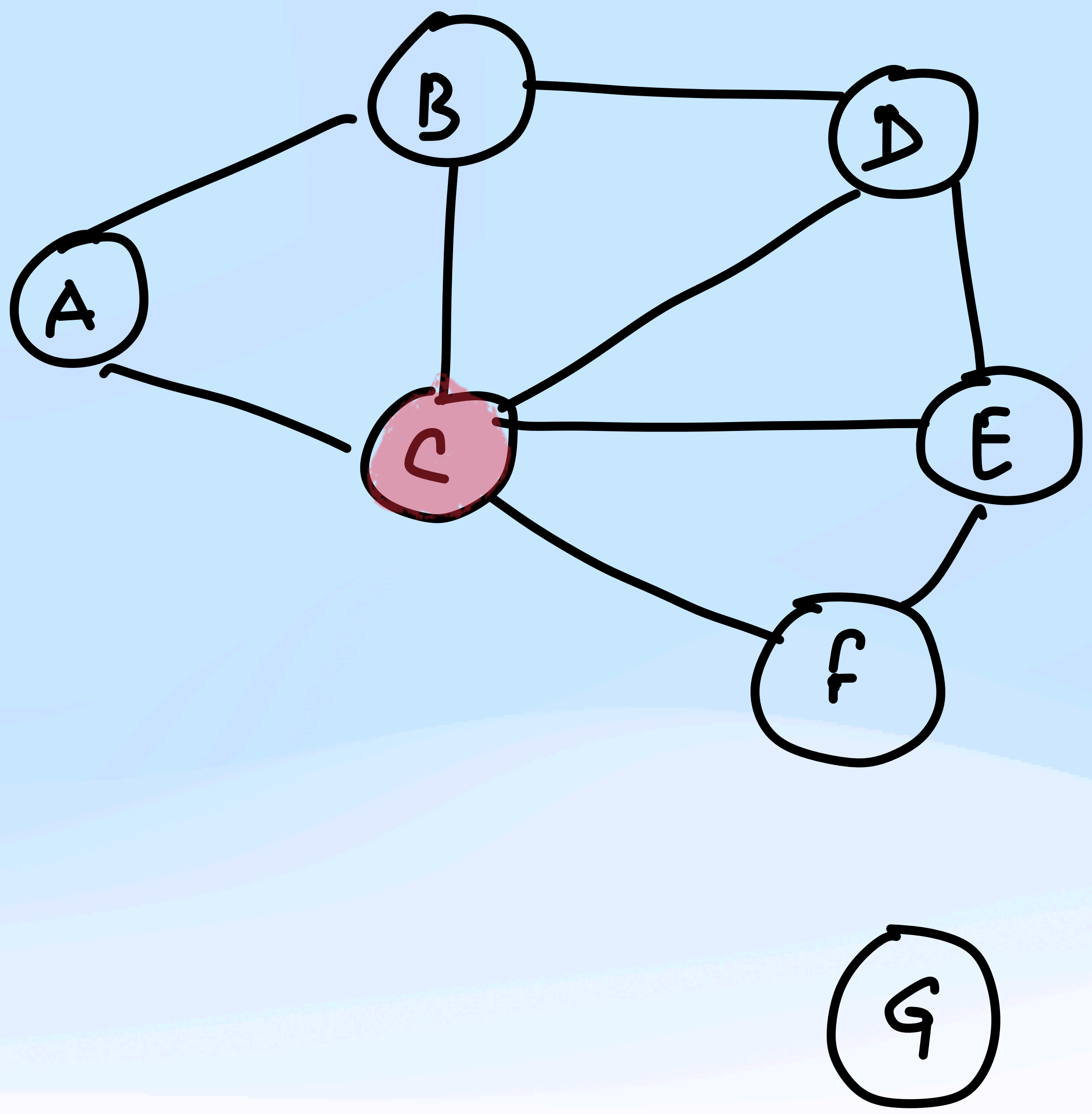
Possible approach:

- Start one queen at a time, place the next queen
- If some queen is not placed, try placing it
- Check goal state



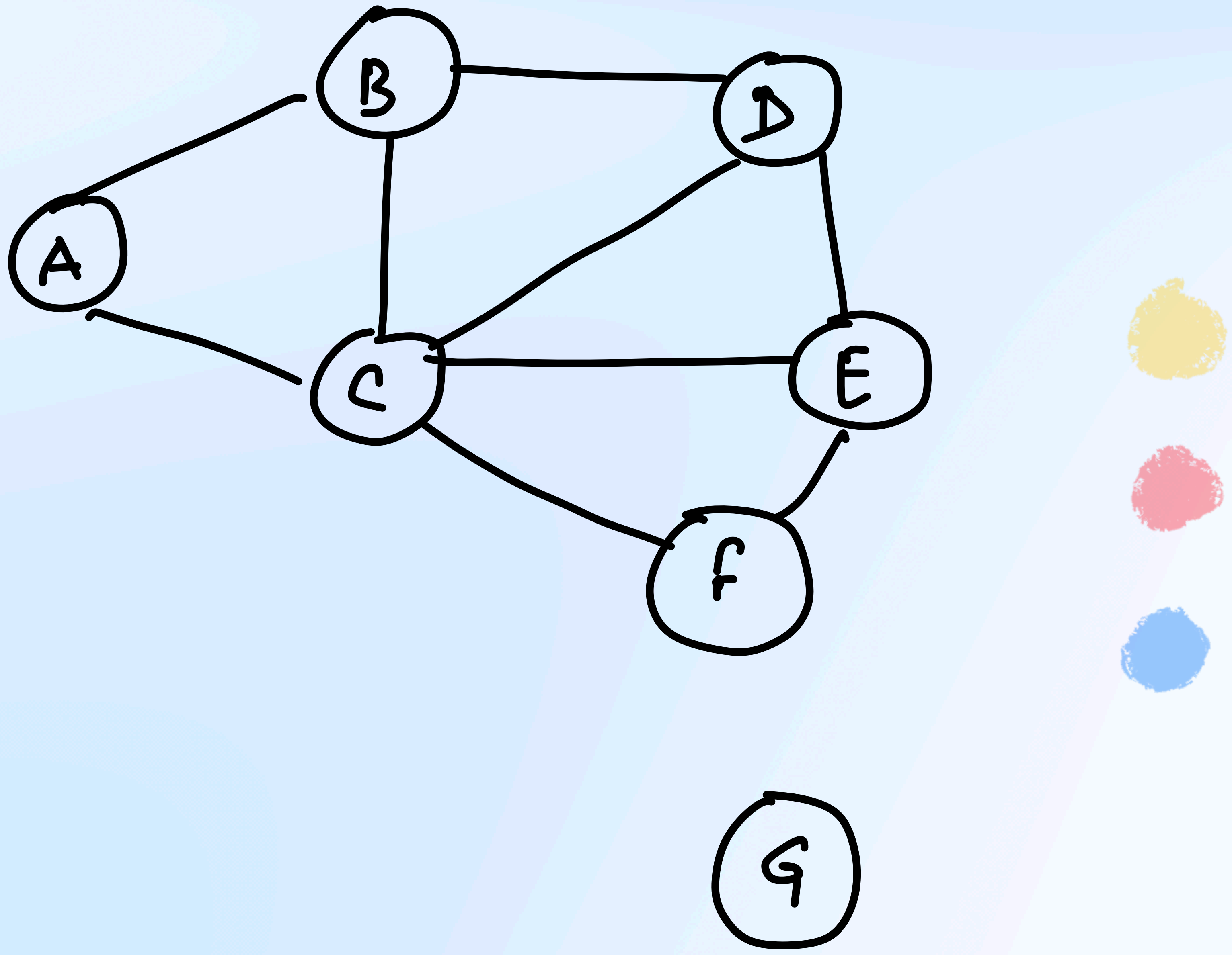


Map colouring problem



A map of a country

- Assign colours to regions so that adjacent regions are having different colours
- Equivalently vertex colouring of a graph



Factored representation and CSP

State

- Set of variables each of which has a value
- Variables have constraints

Goal

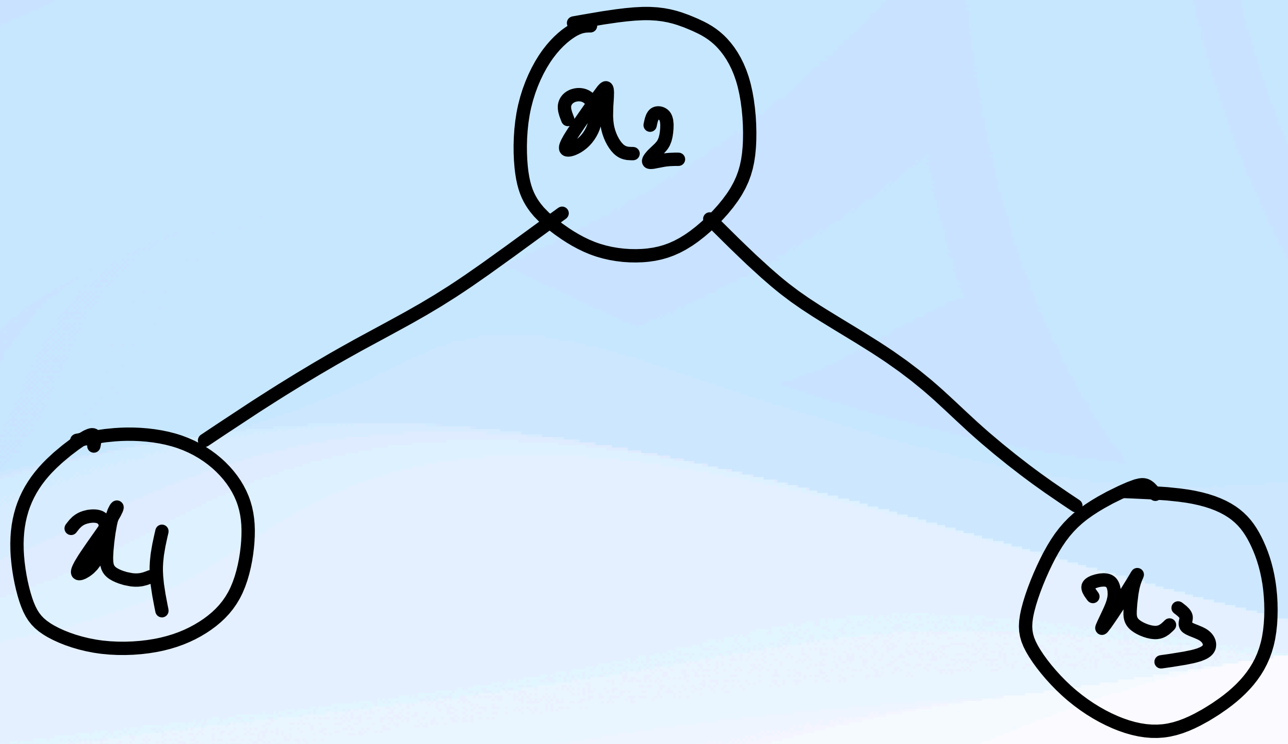
- Assignment of values to all variables such that constraints are satisfied

Constraint satisfaction problem

- A set of variables { x1, x2, …, x_n }
- A set of domains { D1, D2, …, D_n }
- A set of constraints that specify allowable combinations of values

CSP : first example

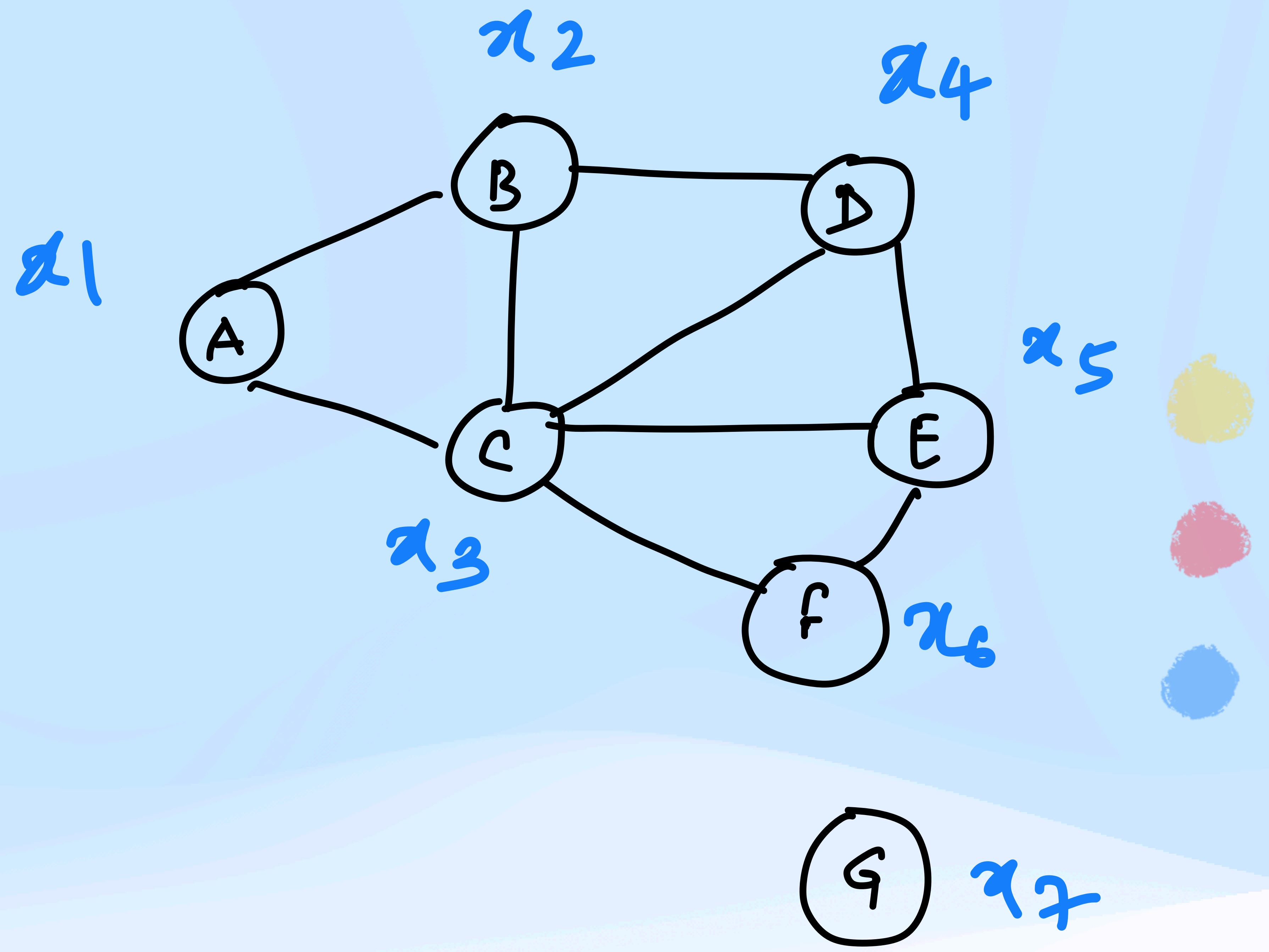
- {x1, x2, x3}
- D1 = D2 = D3 = { 1, 2, 3}
- C12, C23



- **C12 : { <a, b> | a in D1, b in D2, a < b}**
- **C23 : { <a, b> | a in D2, b in D3, a < b}**

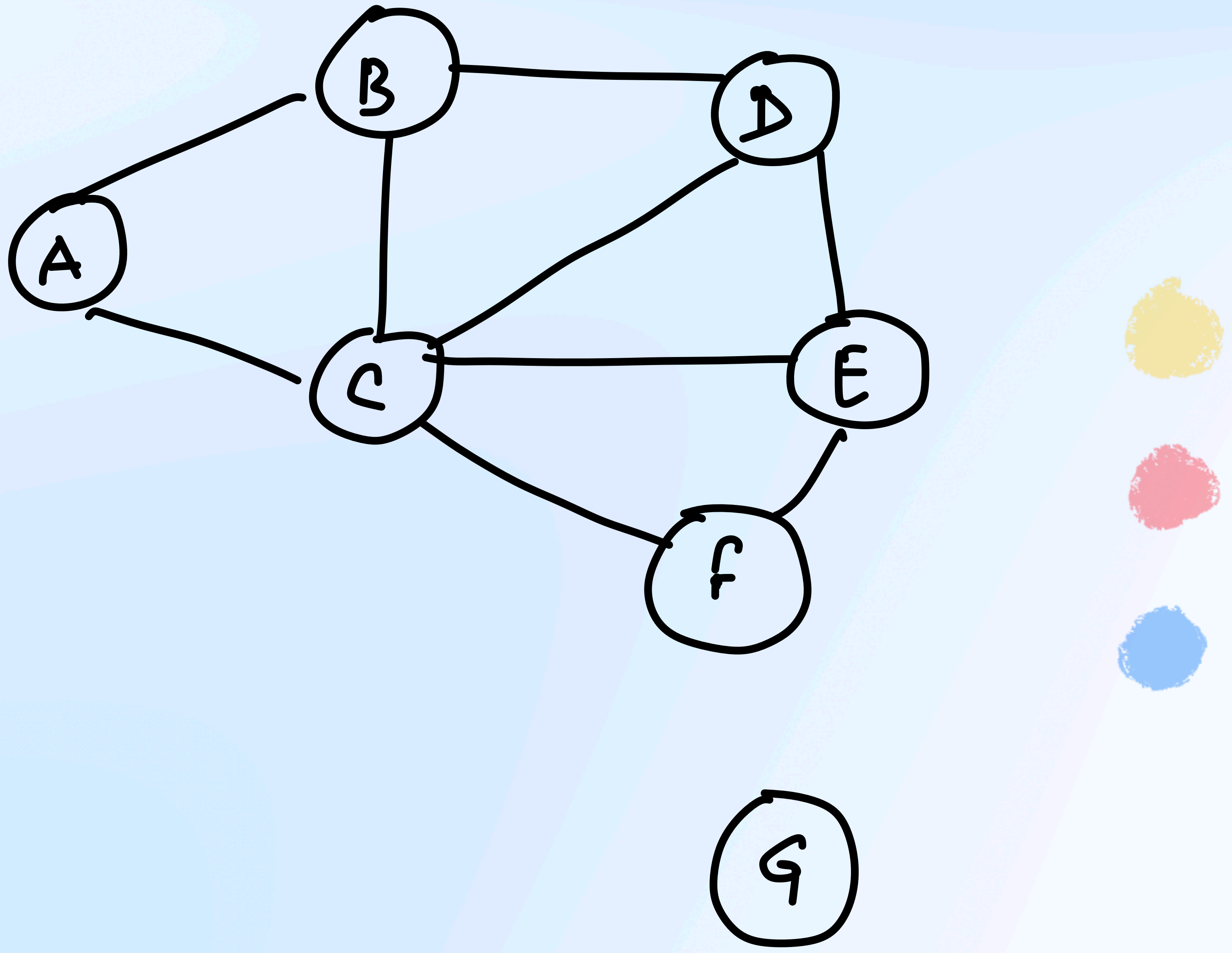
- **C12 : { <1, 2> <1, 3> <2, 3>}**
- **C23 : { <1, 2> <1, 3> <2, 3>}**

Map colouring as CSP



A map of a country

- Assign colours to regions so that adjacent regions are having different colours
- Equivalently vertex colouring of a graph



- **C12 : { <a, b> | a in D1, b in D2, a != b}**
- **C23 : { <a, b> | a in D2, b in D3, a != b}**
- …

N Queens as CSP

Variables : Qa Qb Qc Qd

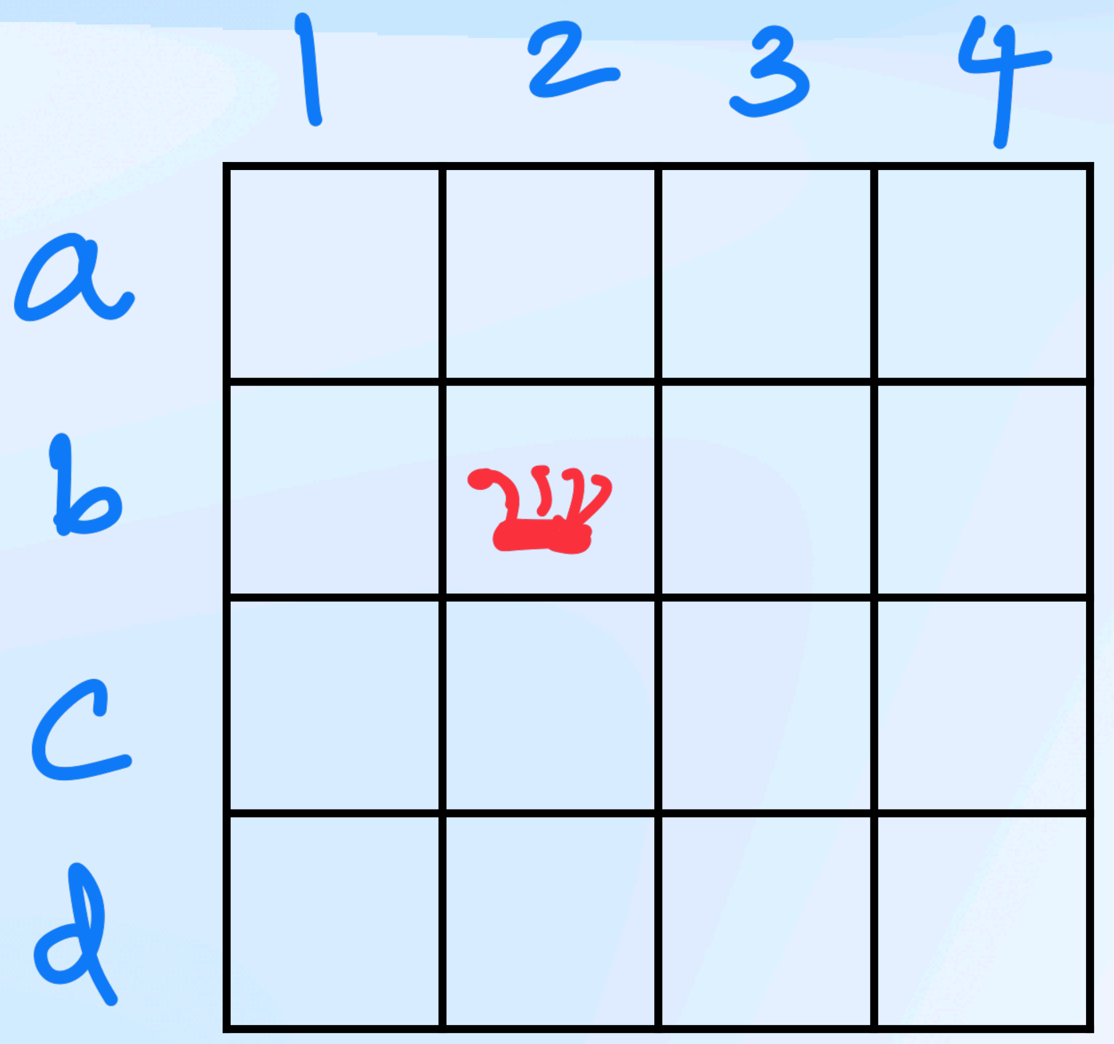
Qa is for row a

Qb is for row b

⋮

N x N empty chess board

- Place N queens such that no queen is under attack by any other queen



$Q_b = 2$

- **C_ab : { <x, y> | x in D1, y in D2, x != y}**
- **C_ac : { <x, y> | x in D2, y in D3, x != y}**
- **…**

if $Q_b = 2$, can $Q_a = 1$?

can $Q_a = 3$?

N Queens as CSP

Variables : Q_a Q_b Q_c Q_d

Q_a is for row a

Q_b is for row b

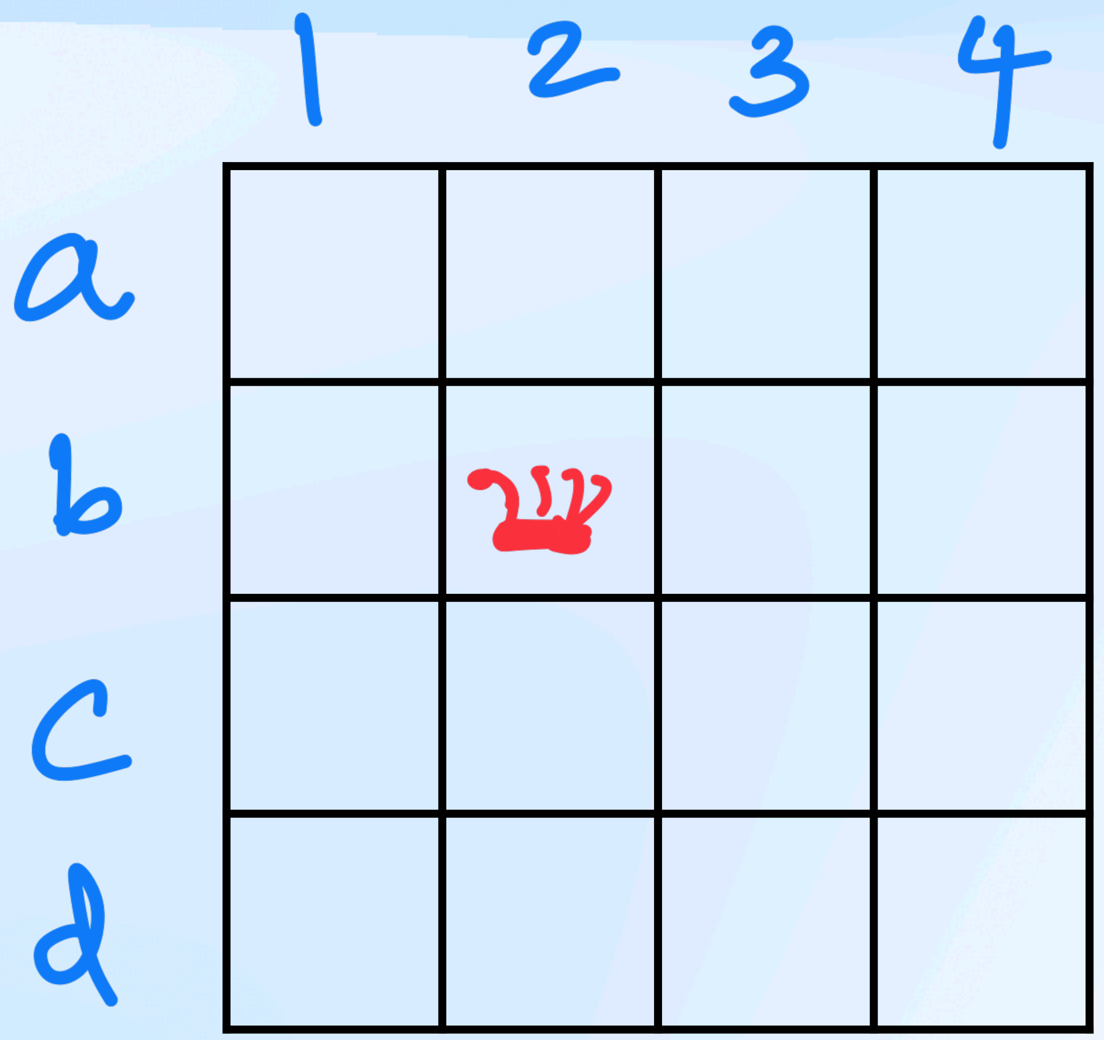
⋮

N x N empty chess board

- Place N queens such that no queen is under attack by any other queen

- **C_ab** : { <x, y> | x in D1, y in D2, x != y}
- **C_ac** : { <x, y> | x in D2, y in D3, x != y}
- ...

and



$$C'_{ab} = \{ \langle x, y \rangle \mid x \text{ in } D_1 , y \text{ in } D_2 \;\; |x-y| \neq |a-b| \}$$

N Queens as CSP

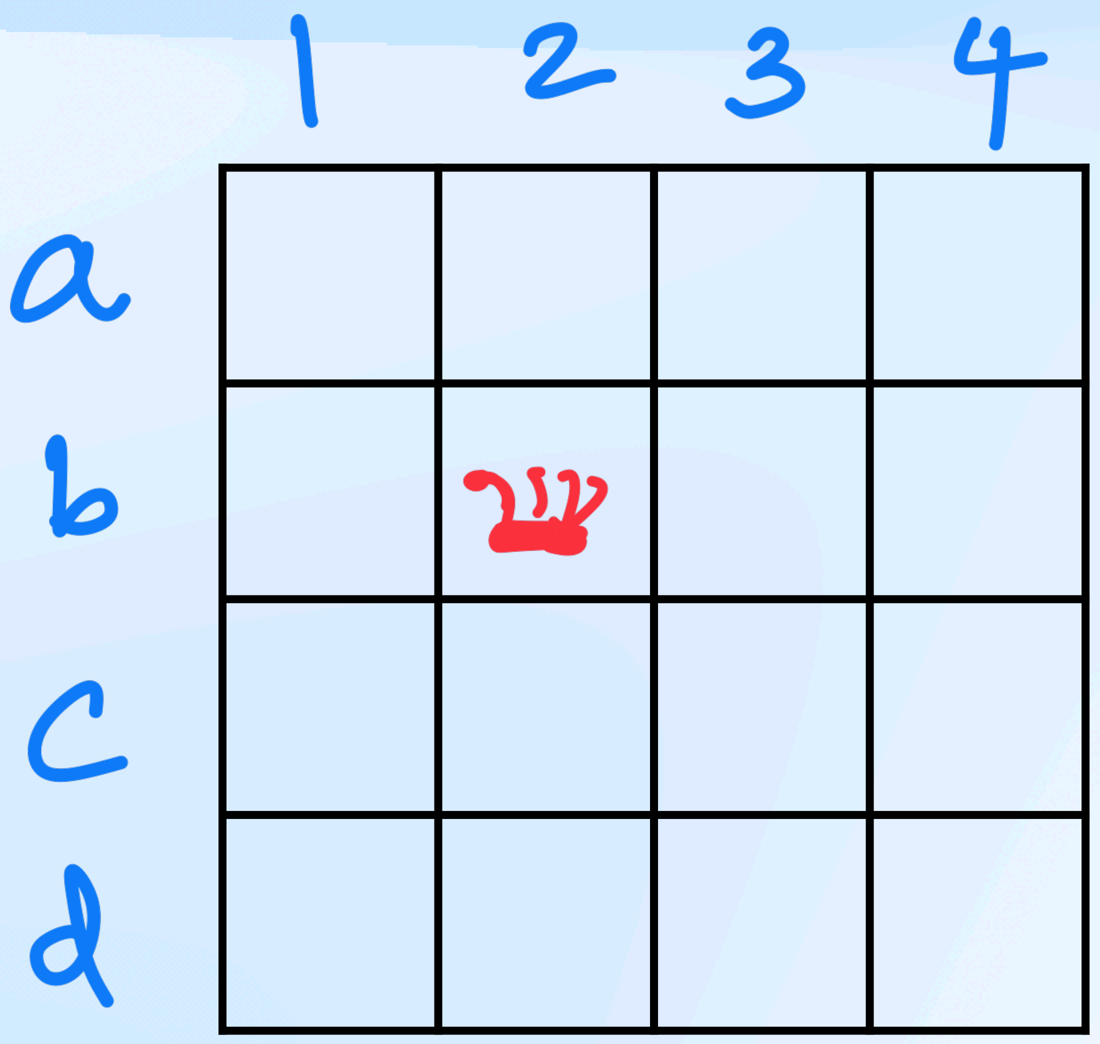
- **C_ab : { <x, y> | x in D1, y in D2, x != y}**

and

$$C'_{ab} = \{ \langle x, y\rangle \mid x \text{ in } D1, y \text{ in } D2, \ |x-y| \neq |a-b| \}$$

N x N empty chess board

- Place N queens such that no queen is under attack by any other queen



- **D1 : {1, 2, 3, 4}** **D2 : {1, 2, 3, 4}**

- **(1,1), (2, 2), (3, 3), (4, 4) :** forbidden by C_ab

- What are entries forbidden by C'_ab?

$$\tilde{C}_{ab} = \left\{ \begin{matrix} \cancel{(1,1)} & \cancel{(1,2)} & (1,3) & (1,4) \\ \cancel{(2,1)} & \cancel{(2,2)} & \cancel{(2,3)} & (2,4) \\ (3,1) & \cancel{(3,2)} & \cancel{(3,3)} & \cancel{(3,4)} \\ (4,1) & (4,2) & \cancel{(4,3)} & \cancel{(4,4)} \end{matrix} \right\}$$

Job scheduling as CSP

We have J1, J2, …, J10

- J1,…,J5 take 2 units of time, others take 5 units each
- J6,…,J10 cannot be started before J1,….,J5 are completed
- J6 and J7 require the same equipment
- All jobs should be completed within 15 units of time.

$J_1 \quad J_2 \ \ldots \quad J_{10}$

$D_i = \{1 .. \ 15\}$

$J_1 + 2 \leq 15$

$\vdots$

$J_6 + 5 \leq 15$

$\vdots$

$J_6 \geq J_1 + 2$

$J_6 \geq J_2 + 2$

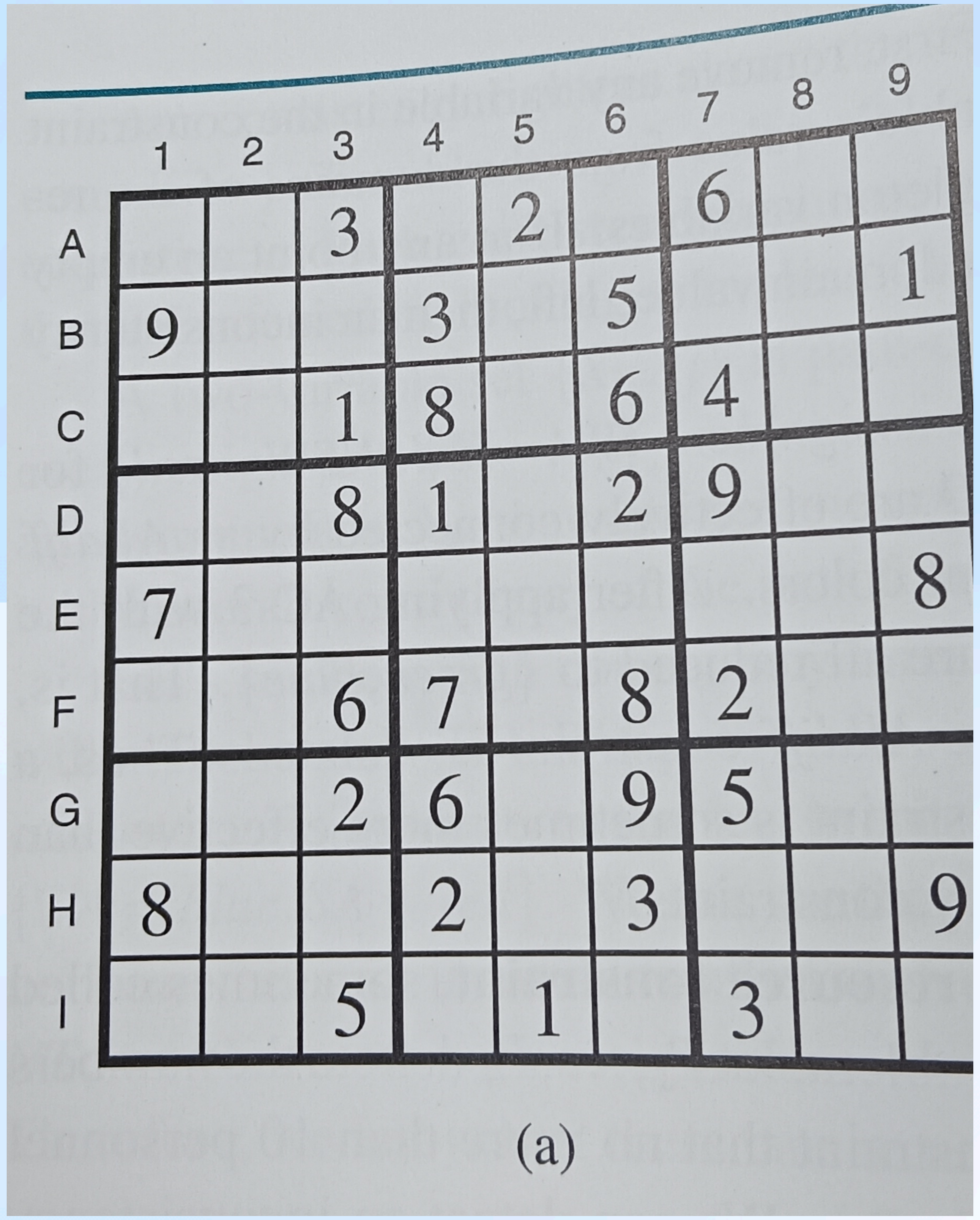
$\vdots$

$J_6 \geq J_7 + 5$

OR

$J_7 \geq J_6 + 5$

Sudoku as CSP



What are the variables, domains, constraints?

Alldiff (A1**,** A2,…, A9)

Alldiff (B1**,** B2,…, B9)

…

Alldiff (A1**,** B1,…, I1)

Alldiff (A2**,** B2,…, I2)

…

Alldiff (A1**,** A2, A3, B1, B2, …, C3)

…

Solving a CSP

State : partial assignment of variables

Action : extend the current assignment by assigning a variable a value from its domain

A DFS like backtracking algorithm

- Which variable should be assigned next? If a variable has multiple values possible, which one should we select?
- Can we backtrack more than once step?
- What inferences can we perform at each step?
    - Node consistency : if all values in the domain of the node satisfy the unary constraints
    - Arc consistency
    - Path consistency (k consistency)

Revising domain for arc consistency

X_i is consistent with respect to X_j if for every value in Di there is some value in Dj that satisfies the binary constraint on the arc (X_i, X_j)

RevisePair (X_i, X_j) X_i ⟶ X_j

// Returns true if domain of X_i is modified; false otherwise.

revised = false

for every a in D_i do

If there does not exist any value b in D_j such that the pair (a, b) satisfies the constraint between X_i, X_j

delete a from D_i ;

revised = true

End if

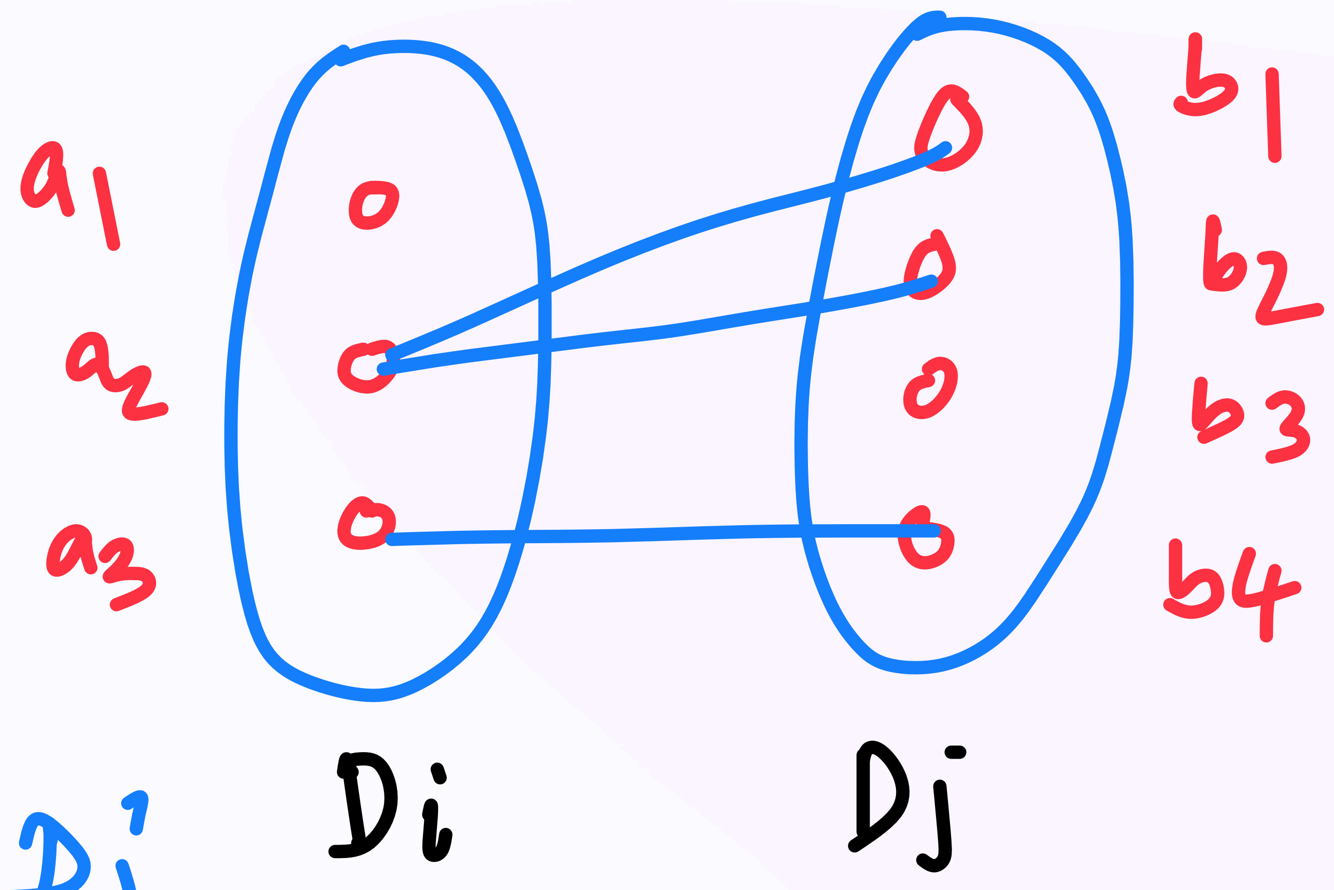
End for

return revised

RevisePair (xi, xj)

- Deletes a1 from Di
- does not delete b3 from Dj

RevisePair (xj, xi) deletes b3 from Dj



Arc consistency

ArcConsistency // **Returns false if inconsistency is found; true otherwise.**

Q = queue of all arcs in the CSP

X_i ⟶ **X_j**

while **Q** is not empty

(X_i, X_j) = remove element from **Q**

If revisePair (X_i, X_j) == true

If D_i is empty return false

For each X_k such that (X_k, X_i) is a relation in CSP where k not equal to j

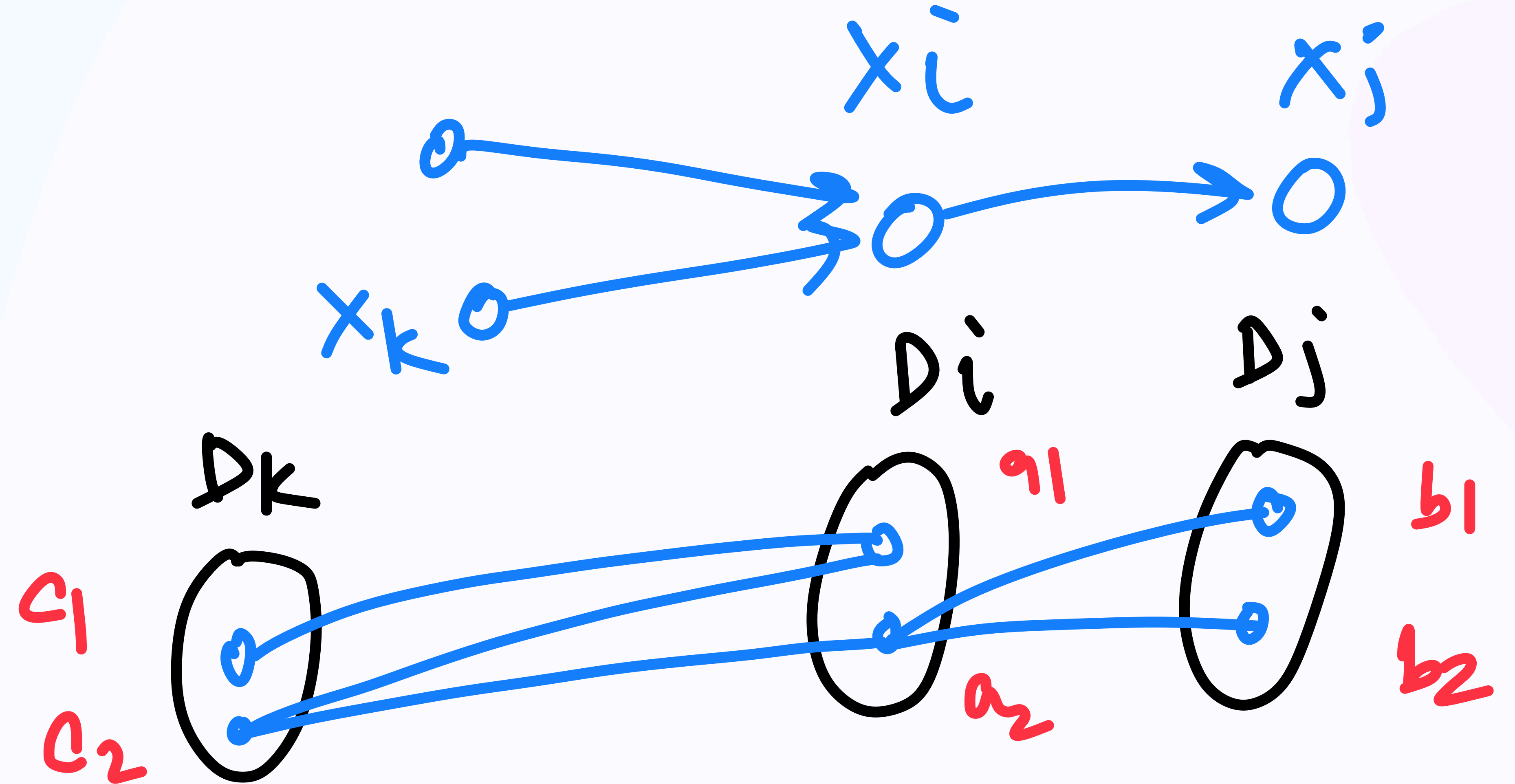
Add (X_k, X_i) to **Q**

End for

End if

End while

Return true



deletion of a_1 from Di causes c_1 to be deleted from Dk

Arc consistency

ArcConsistency // Returns false if inconsistency is found; true otherwise.

Q = queue of all arcs in the CSP

while **Q** is not empty

(X_i, X_j) = remove element from **Q**

If revisePair (X_i, X_j) == true

If D_i is empty return false

For each X_k such that (X_k, X_i) is a relation in CSP where k not equal to j

Add (X_k, X_i) to **Q**

End for

End if

End while

Return true

$X_i \longrightarrow X_j$

Max size of any domain = d

# of arcs = c

complexity of RevisePairs = $O(d^2)$.

complexity of Arc Consistency

- An arc $X_i \to X_j$ is added to Q again if a value from D_j is removed.

# of times arc is added to Q = $O(d)$.

Time Complexity of ArcConsis. = $O(cd^3)$.

Structured CSPs

What if the Constraint graph has a structure?

simplest case: constraint graph is a **tree**. In this case the CSP is efficiently solvable.

$O(n \cdot d^2)$ time.

TreeCSP : // returns a solution or a failure

n : number of variables in CSP

assignment = empty assignment

Root the constraint graph at an arbitrary node (variable)

X_1, X_2, … X_n is an order of variables where parent appears before node for each node

For i = n downto 2 do

note the order, it is from leaves up to root.

RevisePair (parent (X_i), X_i); If inconsistency found return false

End for

For i = 1 to n do

this is from root to leaves.

assignment(X_i) = any consistent value from D_i

If no consistent value found return false

End for

Return assignment

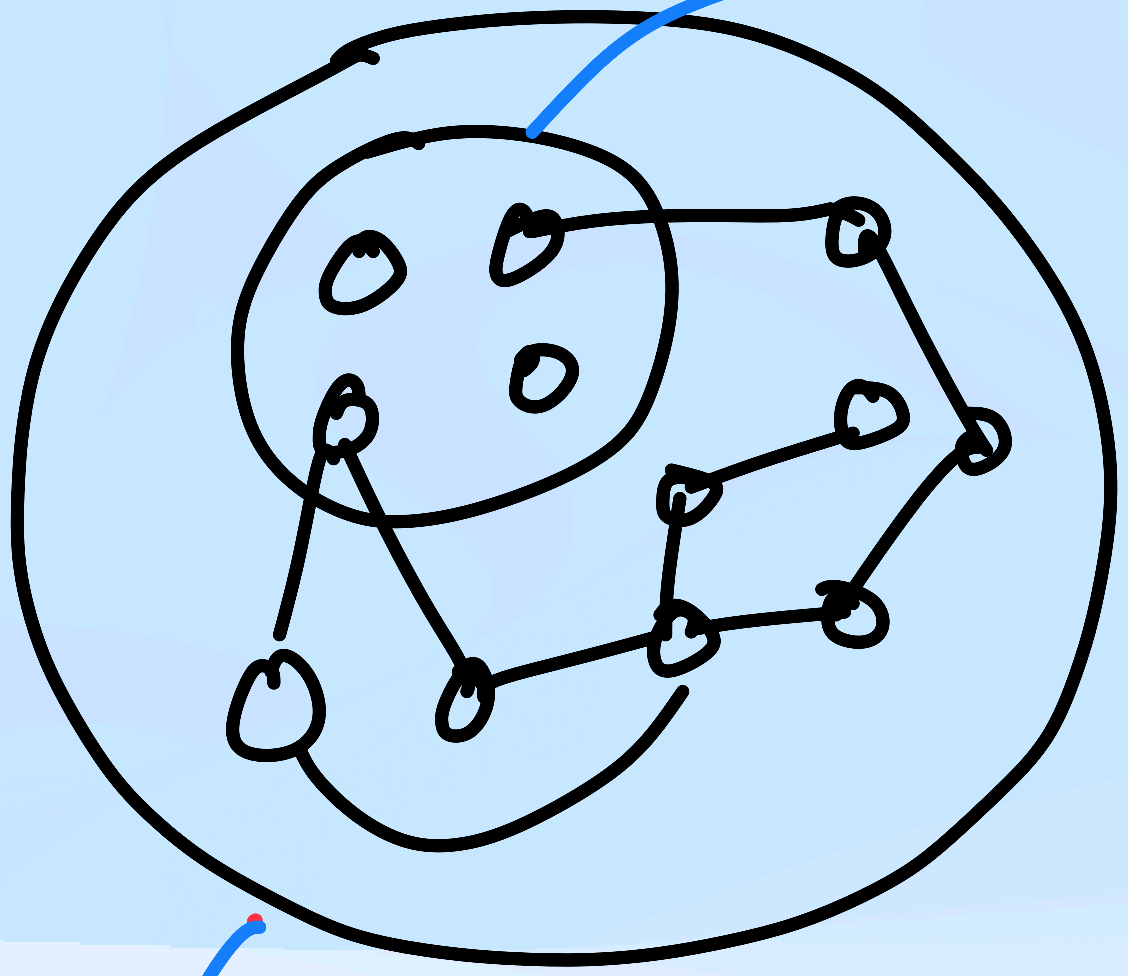
Why does it work for trees? Does it work for Directed Acyclic Graphs?

— where does it fail?

Structured CSPs

What if the Constraint graph has a structure?

Tree like graphs: use idea of cycle cutset



S: cycle cutset
"if removal of nodes in S leaves an acyclic subgraph."

constraint graph

If S is cycle cutset.
Time taken for solving CSP =

$$O\Big(\underbrace{d^{|S|}}_{\text{bruteforce on } |S|} \cdot \underbrace{(n-|S|)\cdot d^2}_{\text{Tree CSP solve on } (n-|S|) \text{ variables.}}\Big)$$

- cycle cutset may not be small always.
- minsize cycle cutset is hard to compute.
- can get good approx. to minsize cycle cutset.