Outline

- Introduction
- Constraint propagation
- Backtracking search
- Local search

Outline

- Introduction
- Constraint propagation
- Backtracking search
    - branching strategies
    - constraint propagation
    - heuristics for variable and value ordering
- Local search

Solving CSPs: Formulation as a search problem

- Initial state/node:

  empty assignment: {}

- Actions/arcs:

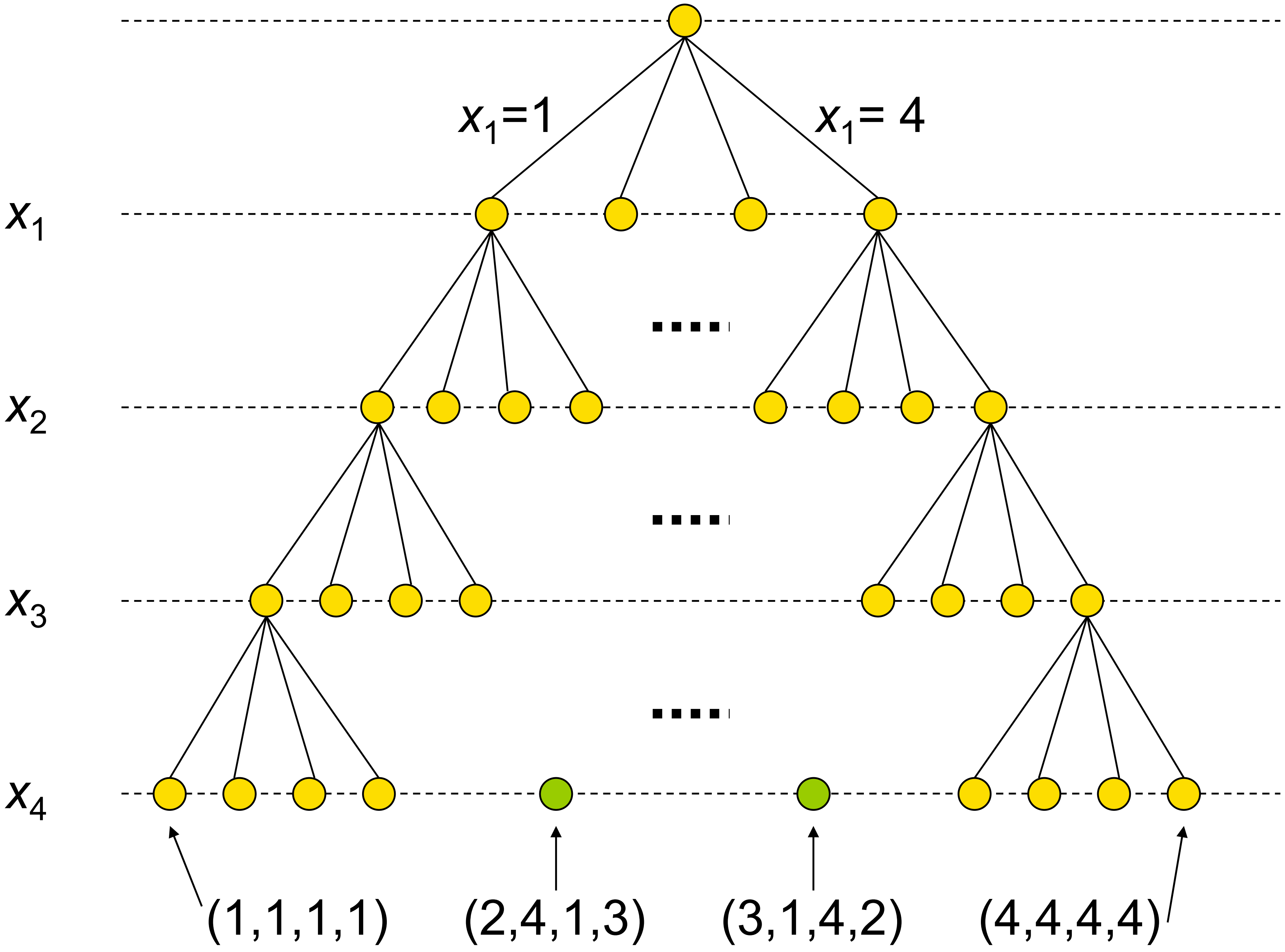
  assign a value to the next variable

- Goal condition:

  current assignment is complete (all variables have a value), and

  assignment satisfies all constraints

Search tree for 4-queens



Which search algorithm?

- Breadth-first search?
- Depth-first search?
- Depth-first search with iterative deepening?

Backtracking search

- A backtracking search is a depth-first traversal of a *search tree*
  - search tree is generated as the search progresses
  - search tree represents alternative choices that may have to be examined in order to find a solution
  - method of extending a node in the search tree is often called a *branching strategy*

Popular branching strategies

- Let x be the variable branched on, let $dom(x) = \{1, \ldots, 6\}$
- Enumeration (or *d*-way branching)
  - variable x is instantiated in turn to each value in its domain
  - e.g., $x = 1$ is posted along the first branch, $x = 2$ along second branch, …
- Binary choice points (or 2-way branching)
  - variable x is instantiated to some value in its domain
  - e.g., $x = 1$ is posted along the first branch, $x \neq 1$ along second branch, respectively
- Domain splitting
  - constraint posted splits the domain of the variable
  - e.g., $x \leq 3$ is posted along the first branch, $x > 3$ along second branch, respectively

Backtracking search algorithm template

```
backtrack( assignment, csp )
    if assignment is complete then solution found
    $var \leftarrow$ select next variable
    for each value in dom( var ) do
        save( csp )
        $assignment \cup \{var = value\}$
        if propagate( assignment, csp ) then
            backtrack( assignment, csp )
        endif
        restore( csp )
    endfor
```

CSP for 4-queens

variables:

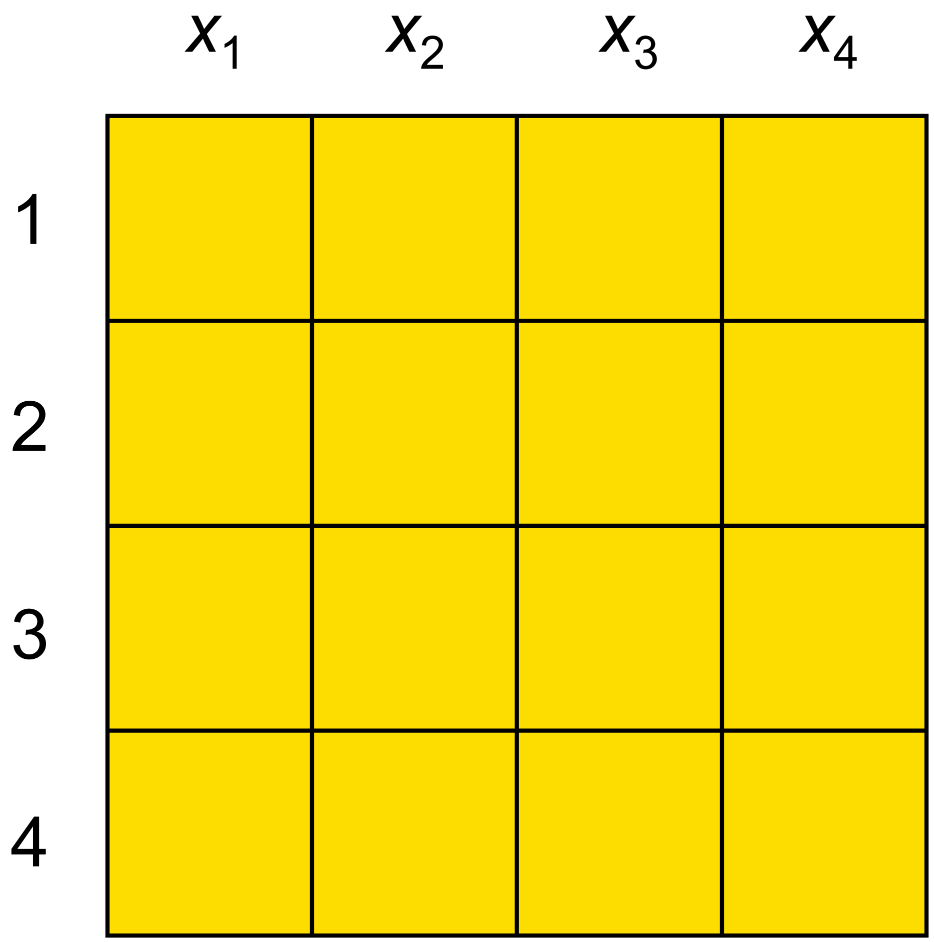
x_1, x_2, x_3, x_4

domains:

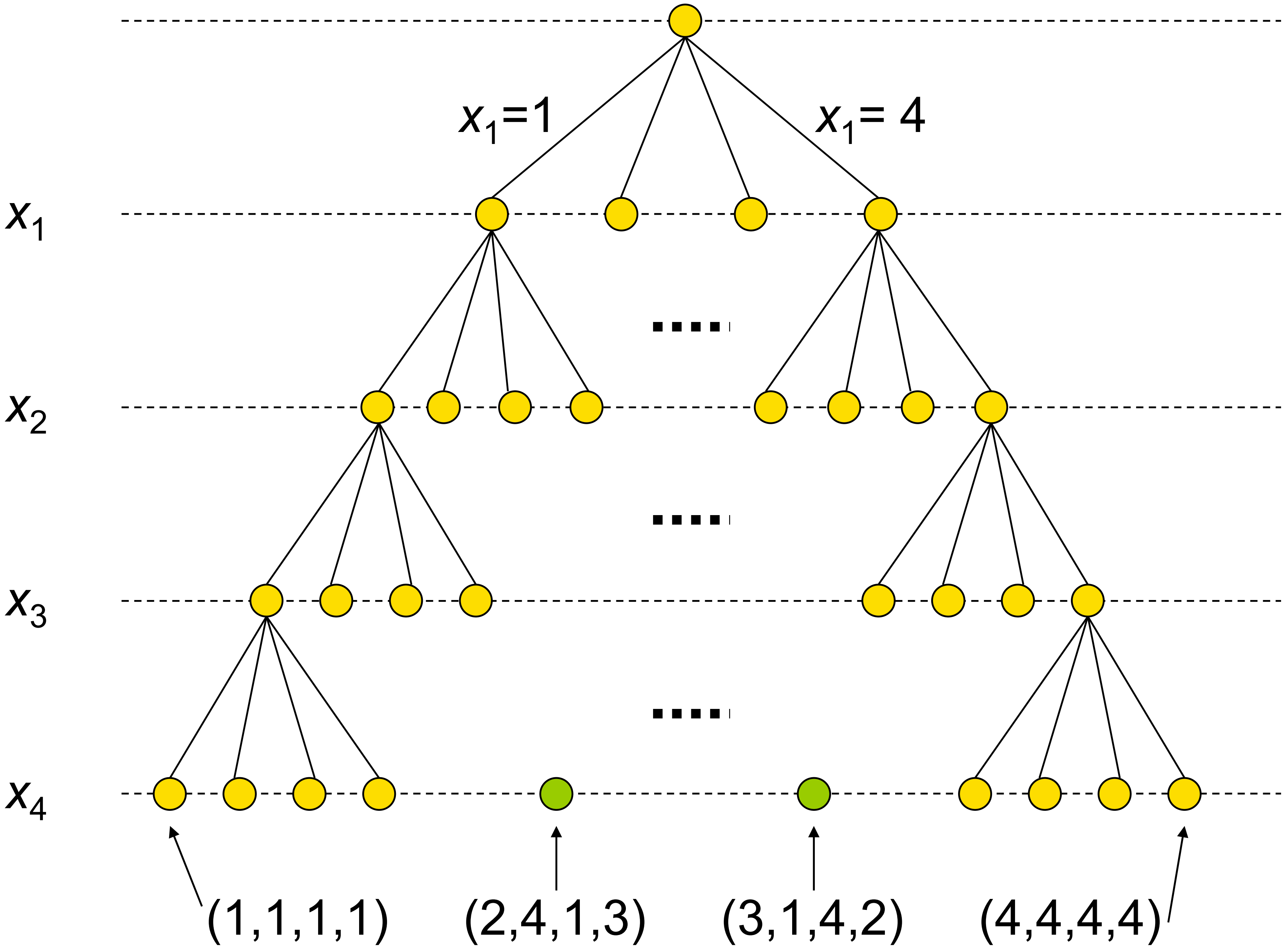
$\{1, 2, 3, 4\}$

constraints:

$$
\begin{aligned}
x_1 &\neq x_2 \quad \wedge \quad |x_1 - x_2| \neq 1 \\
x_1 &\neq x_3 \quad \wedge \quad |x_1 - x_3| \neq 2 \\
x_1 &\neq x_4 \quad \wedge \quad |x_1 - x_4| \neq 3 \\
x_2 &\neq x_3 \quad \wedge \quad |x_2 - x_3| \neq 1 \\
x_2 &\neq x_4 \quad \wedge \quad |x_2 - x_4| \neq 2 \\
x_3 &\neq x_4 \quad \wedge \quad |x_3 - x_4| \neq 1
\end{aligned}
$$



Search tree for 4-queens



Outline

- Introduction
- Constraint propagation
- Backtracking search
  - branching strategies
  - constraint propagation
  - heuristics for variable ordering
- Local search

Constraint propagation

- Effective backtracking algorithms for CSPs maintain a level of local consistency during the search; i.e., perform constraint propagation

- Perform constraint propagation at each node in the search tree
    - if any domain of a variable becomes empty, inconsistent so backtrack (and undo any of the effects of the most recent constraint propagation)

Constraint propagation

- Backtracking search integrated with constraint propagation has two important benefits
  1. removing inconsistencies during search can dramatically prune the search tree by removing deadends and by simplifying the remaining sub-problem
  2. some of the most important variable ordering heuristics make use of the information gathered by constraint propagation

Some backtracking algorithms

- Naïve backtracking (BT)
  - performs no constraint propagation, only checks a constraint if all of its variables have been instantiated; chronologically backtracks
- Forward checking (FC)
  - maintains arc consistency on all constraints with exactly one uninstantiated variable; chronologically backtracks
- Maintaining arc consistency (MAC)
  - maintains arc consistency on all constraints with at least one uninstantiated variable; chronologically backtracks

CSP for 4-queens

variables:

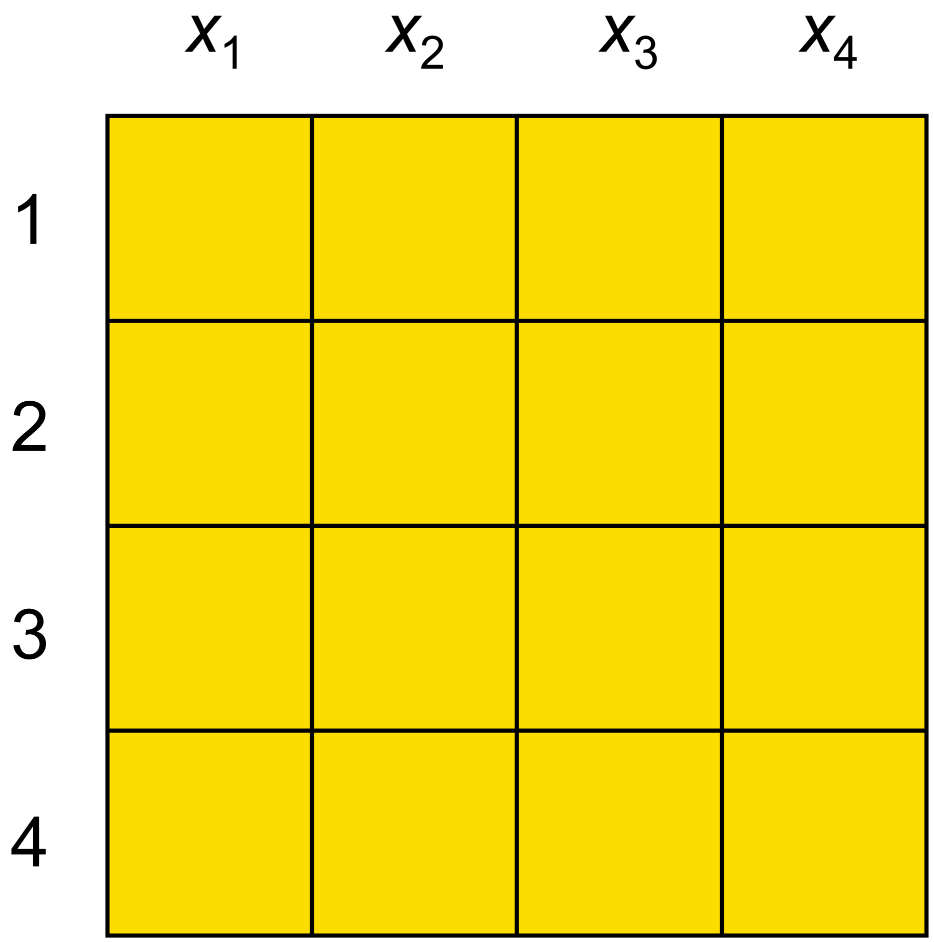
x_1, x_2, x_3, x_4

domains:

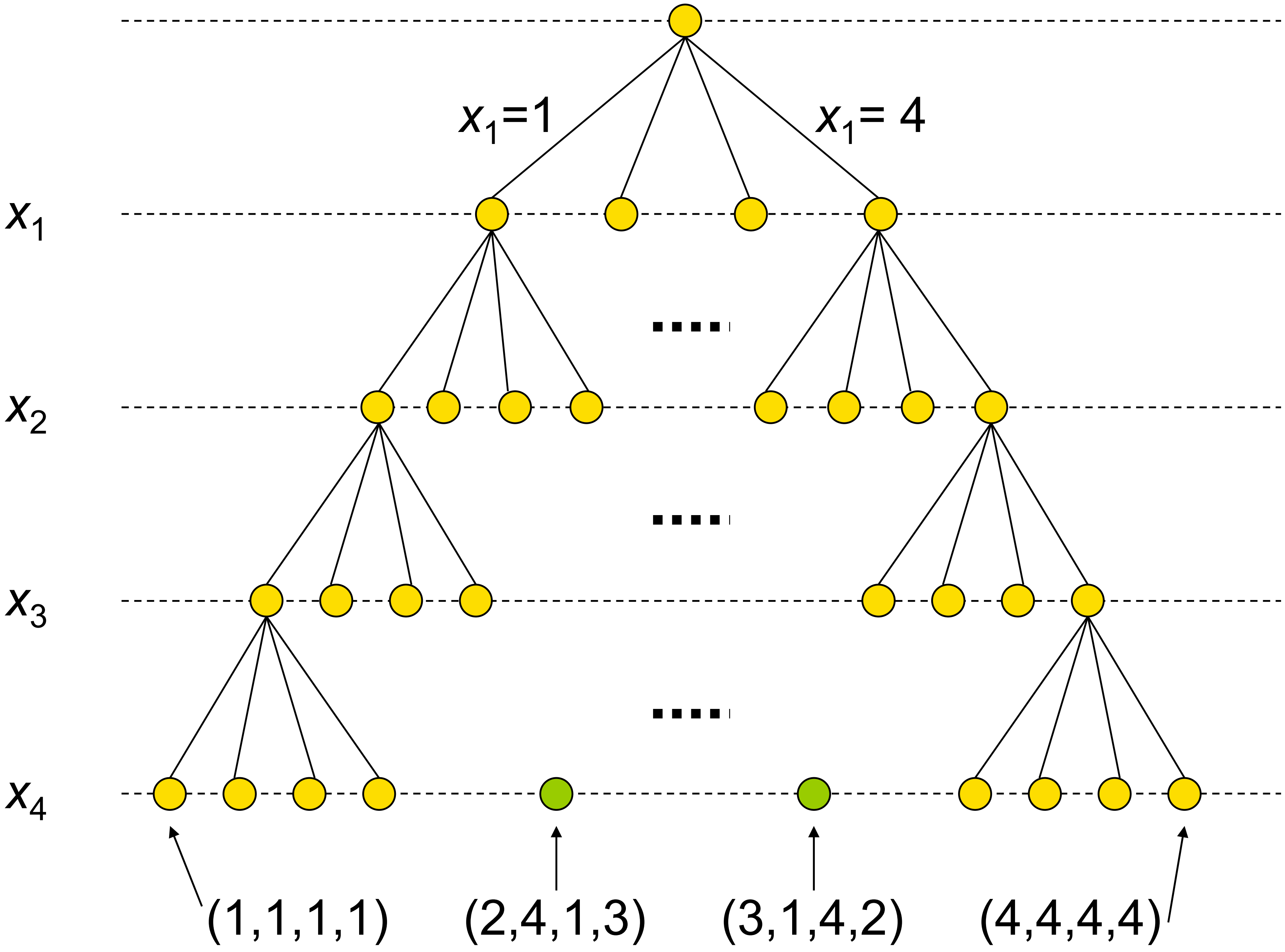
$\{1, 2, 3, 4\}$

constraints:

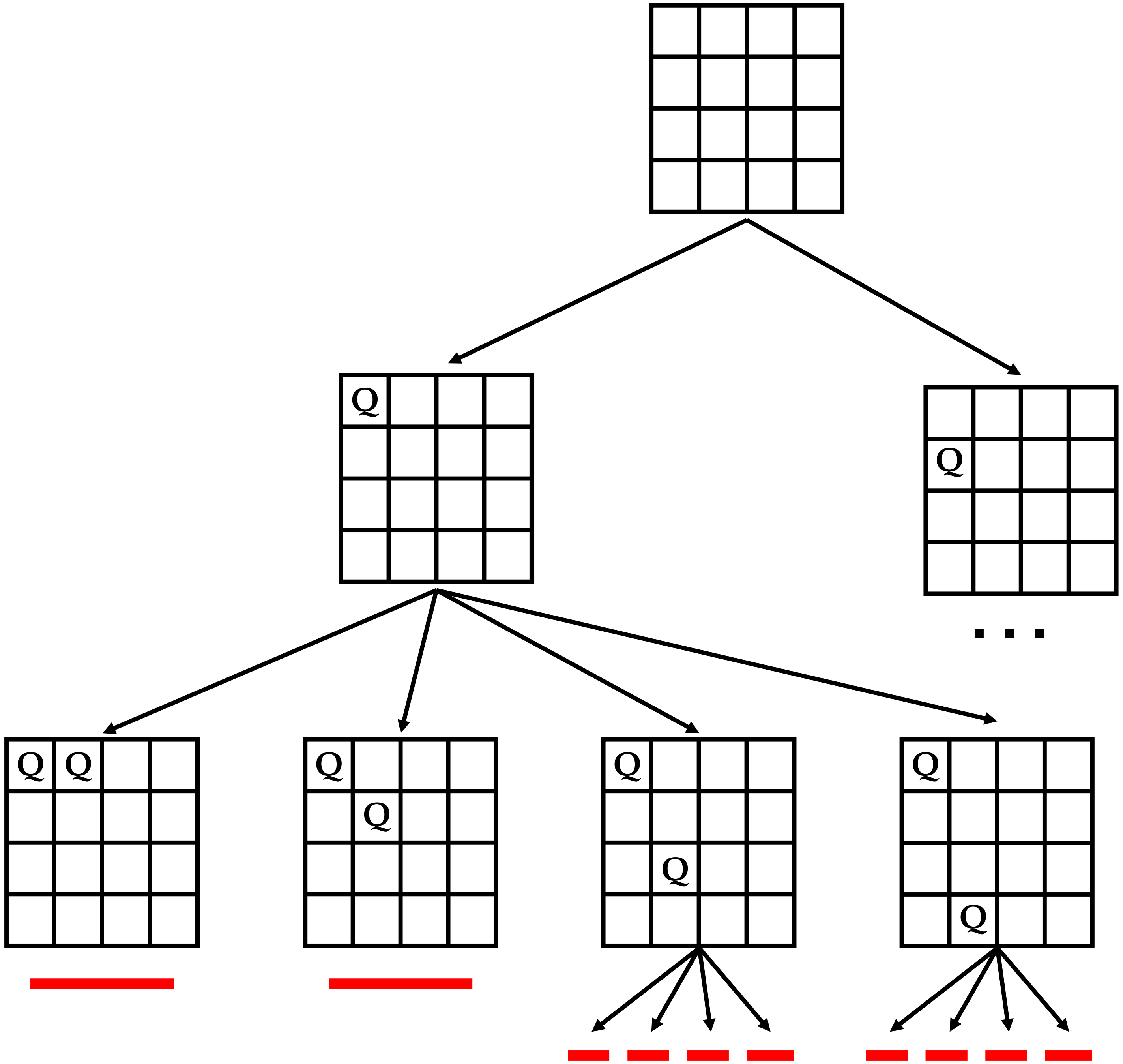
$$\begin{aligned}
x_1 &\neq x_2 \quad \wedge \quad |x_1 - x_2| \neq 1\\
x_1 &\neq x_3 \quad \wedge \quad |x_1 - x_3| \neq 2\\
x_1 &\neq x_4 \quad \wedge \quad |x_1 - x_4| \neq 3\\
x_2 &\neq x_3 \quad \wedge \quad |x_2 - x_3| \neq 1\\
x_2 &\neq x_4 \quad \wedge \quad |x_2 - x_4| \neq 2\\
x_3 &\neq x_4 \quad \wedge \quad |x_3 - x_4| \neq 1
\end{aligned}$$



Search tree for 4-queens



Naïve backtracking (BT)



these eventually fail, not shown on slides

Forward checking (FC)

Enforce arc consistency on constraints with exactly one variable uninstantiated

$\{ x_1 = 1\}$

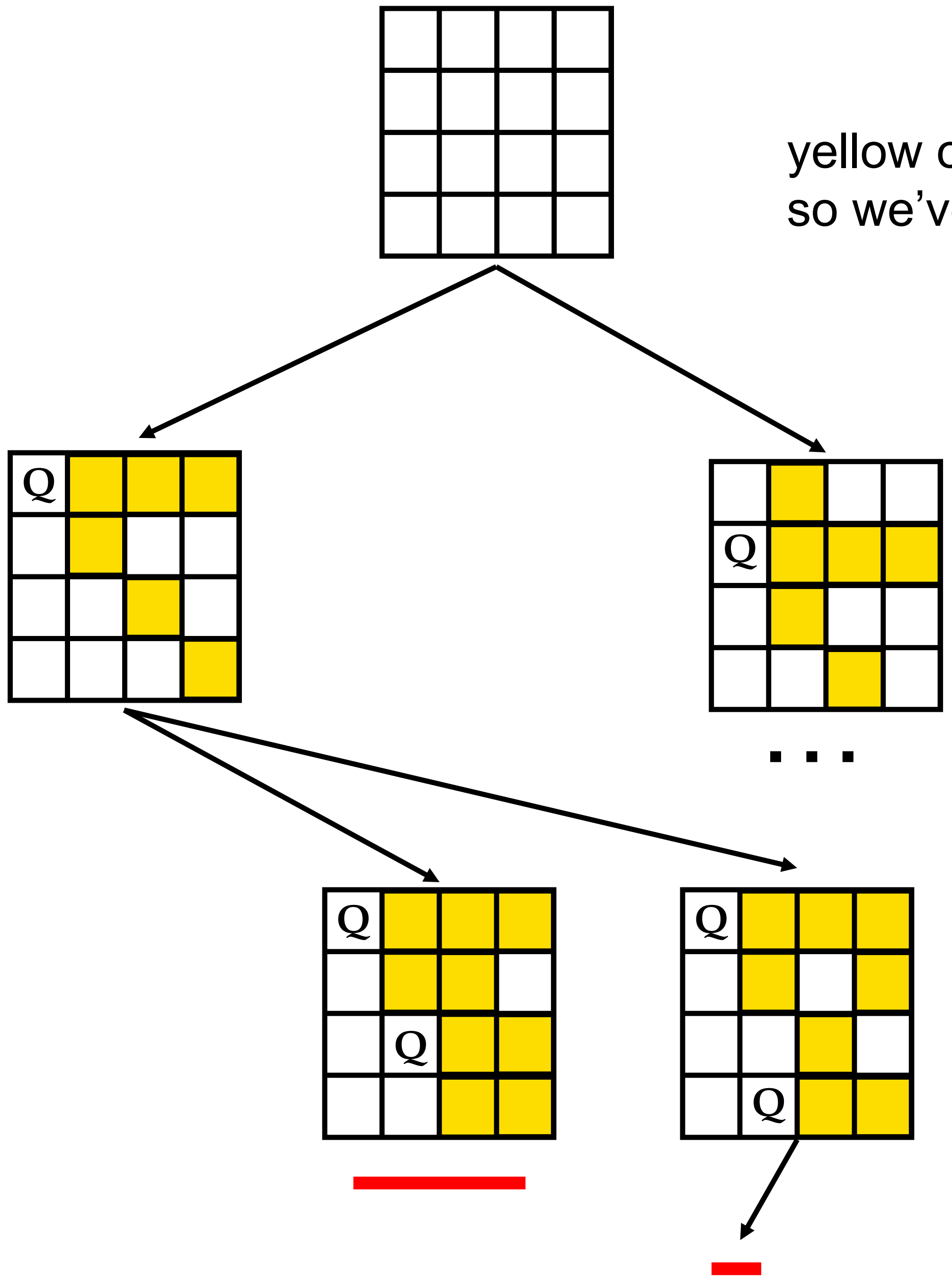
constraints:

$$
\begin{aligned}
&x_1 \neq x_2 \wedge |x_1 - x_2| \neq 1\\
&x_1 \neq x_3 \wedge |x_1 - x_3| \neq 2\\
&x_1 \neq x_4 \wedge |x_1 - x_4| \neq 3
\end{aligned}
$$

	x_1	x_2	x_3	x_4
1	Q	🟨	🟨	🟨
2		🟨		
3			🟨	
4				🟨

yellow ones can’t be done
so we’ve shrunk domains of variables

Forward checking (FC) on 4-queens



yellow ones can’t be done
so we’ve shrunk domains of variables

Maintaining arc consistency (MAC)

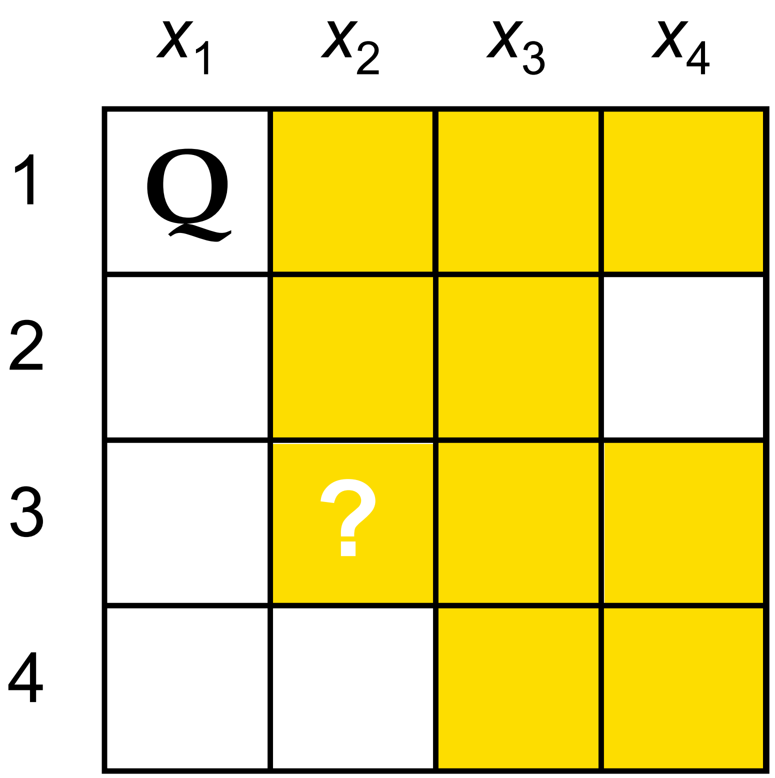
Enforce arc consistency on constraints with at least one variable uninstantiated

$\{ x_1 = 1\}$

constraints:

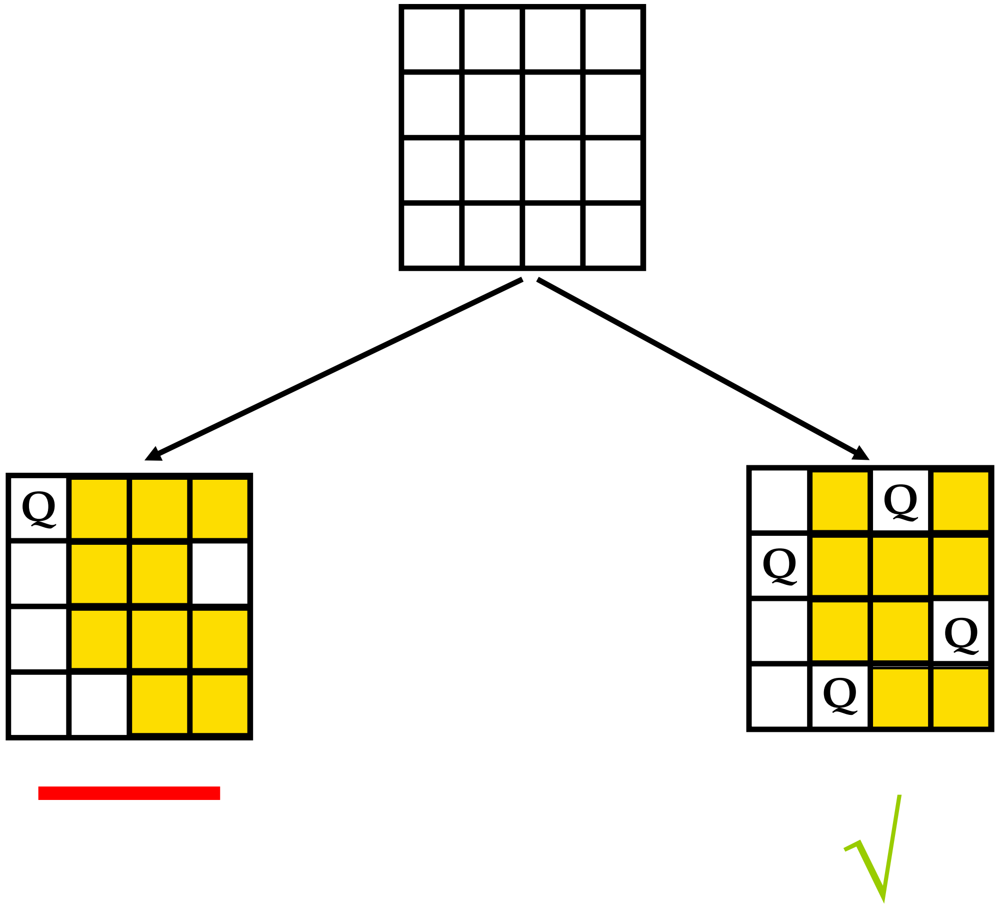
$$x_1 \neq x_2 \quad \wedge \quad | x_1 - x_2 | \neq 1$$
$$x_1 \neq x_3 \quad \wedge \quad | x_1 - x_3 | \neq 2$$
$$x_1 \neq x_4 \quad \wedge \quad | x_1 - x_4 | \neq 3$$
$$x_2 \neq x_3 \quad \wedge \quad | x_2 - x_3 | \neq 1$$
$$x_2 \neq x_4 \quad \wedge \quad | x_2 - x_4 | \neq 2$$
$$x_3 \neq x_4 \quad \wedge \quad | x_3 - x_4 | \neq 1$$

we add these three too



after we do the ones from FC
? can't be, because it removes rest of X3 possibilities

Maintaining arc consistency (MAC) on 4-queens



graph is smaller, but takes more work per node

Outline

- Introduction
- Constraint propagation
- Backtracking search
  - branching strategies
  - constraint propagation
  - heuristics for variable ordering
- Local search

Variable ordering: Basic idea

- Assign a heuristic value to a variable that estimates how difficult it is to find a satisfying value for that variable
- Principle: most likely to fail first
  - *or* don’t postpone the hard part
- Examples:
  - **dom**: choose the variable *x* with the smallest domain size
  - **dom / deg**: divide domain size of a variable by degree of the variable