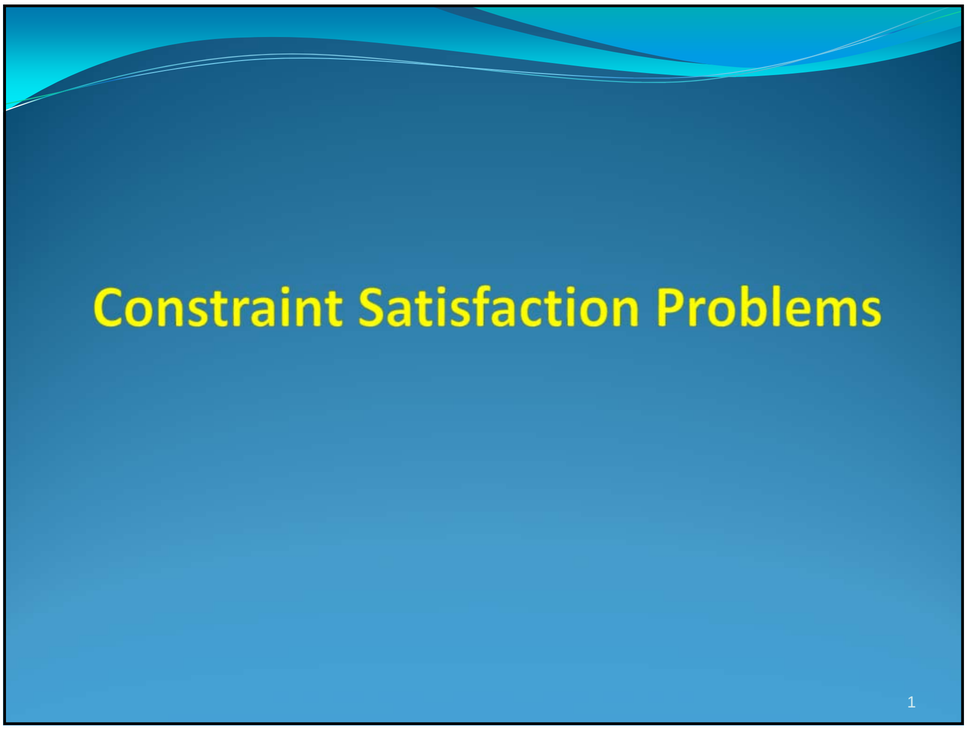
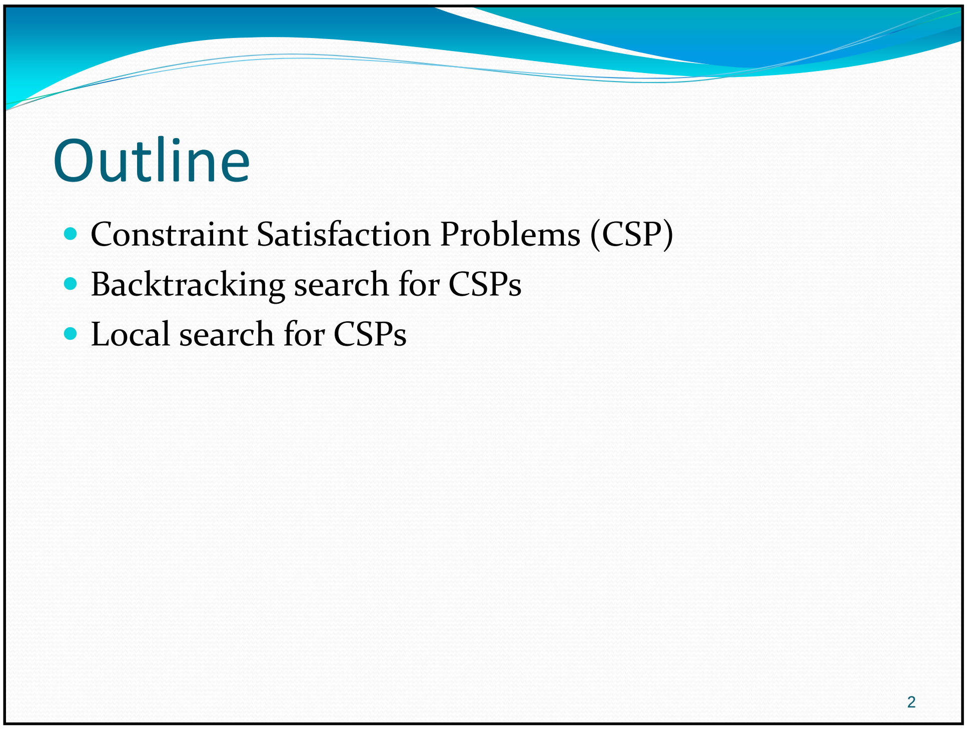
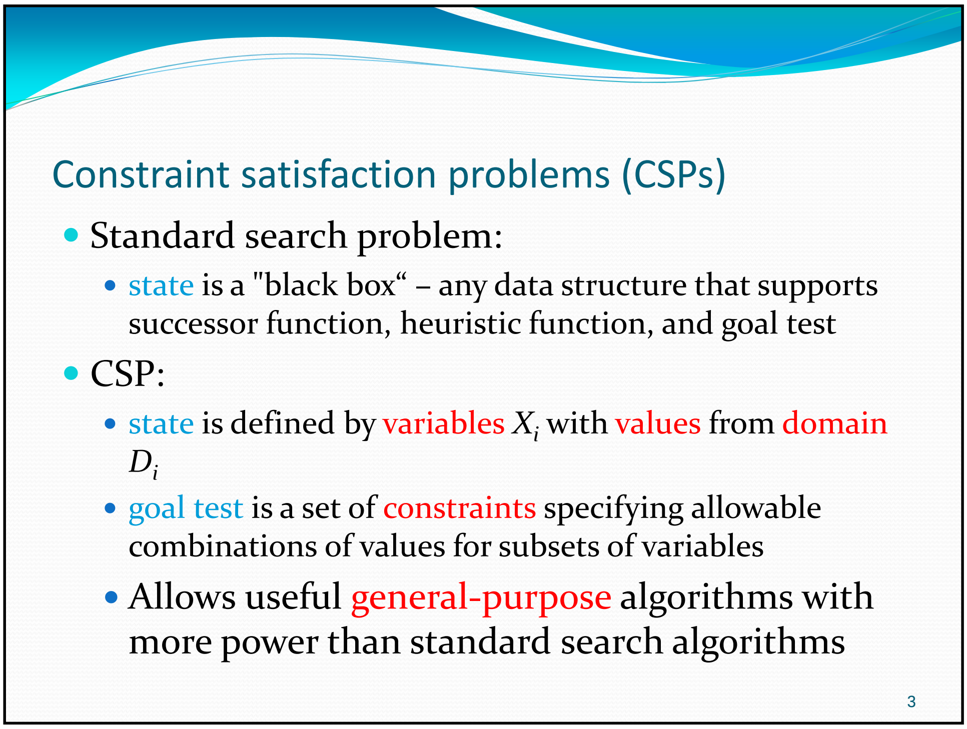


Constraint Satisfaction Problems



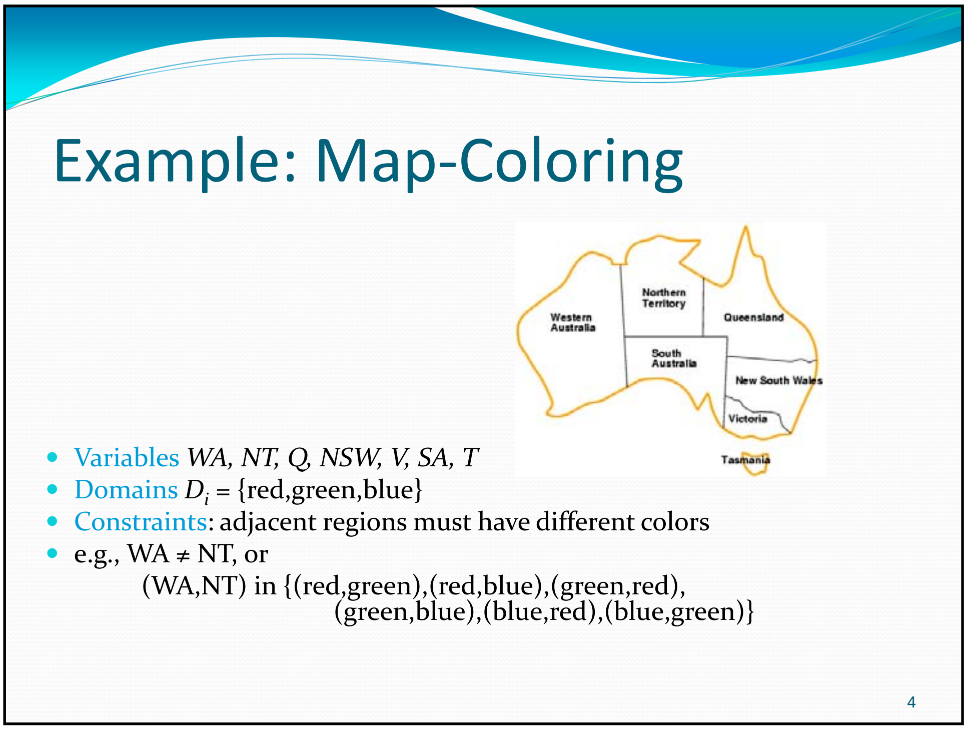
Outline

- Constraint Satisfaction Problems (CSP)
- Backtracking search for CSPs
- Local search for CSPs



Constraint satisfaction problems (CSPs)

- Standard search problem:
  - state is a "black box“ – any data structure that supports successor function, heuristic function, and goal test
- CSP:
  - state is defined by variables X_i with values from domain D_i
  - goal test is a set of constraints specifying allowable combinations of values for subsets of variables
  - Allows useful general-purpose algorithms with more power than standard search algorithms

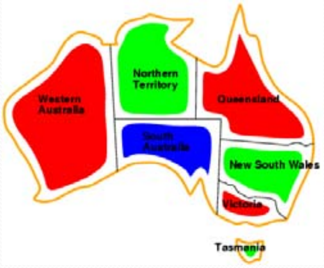


Example: Map-Coloring

Northern Territory
Western Australia
Queensland
South Australia
New South Wales
Victoria
Tasmania

- Variables *WA, NT, Q, NSW, V, SA, T*
- Domains D_i = {red,green,blue}
- Constraints: adjacent regions must have different colors
- e.g., WA ≠ NT, or
  (WA,NT) in {(red,green),(red,blue),(green,red),
  (green,blue),(blue,red),(blue,green)}

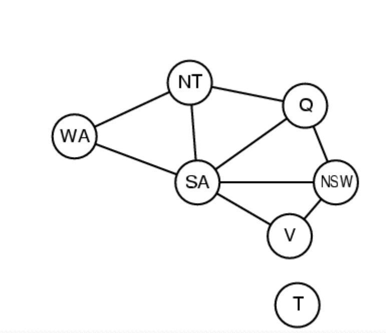
Example: Map-Coloring

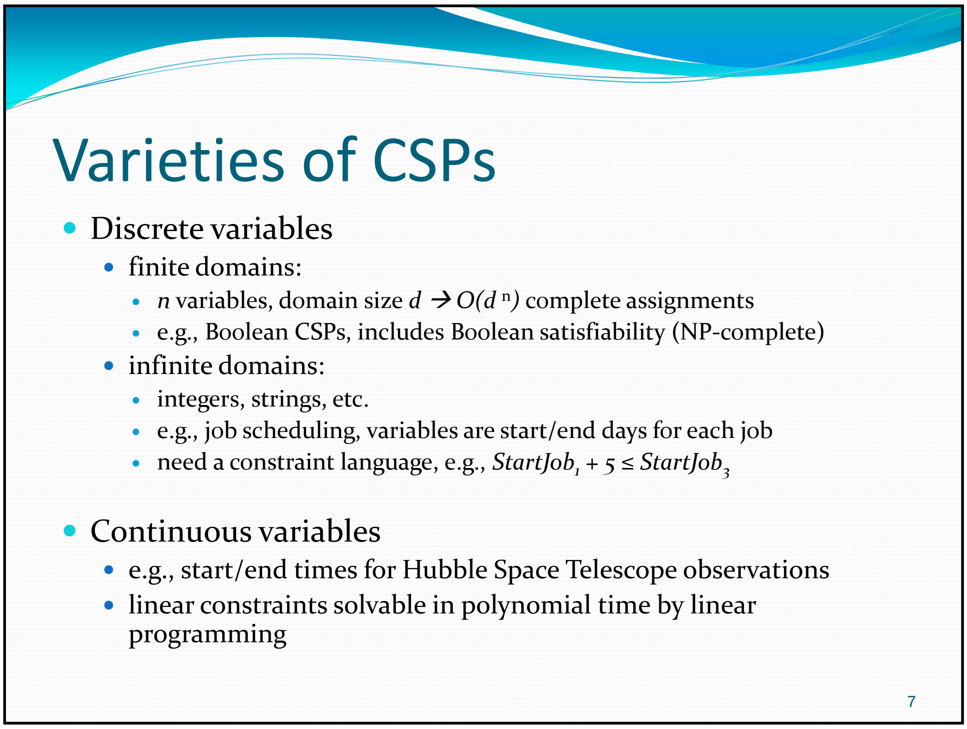


- Solutions are complete and consistent assignments, e.g., WA = red, NT = green, Q = red, NSW = green,V = red, SA = blue, T = green

Constraint graph

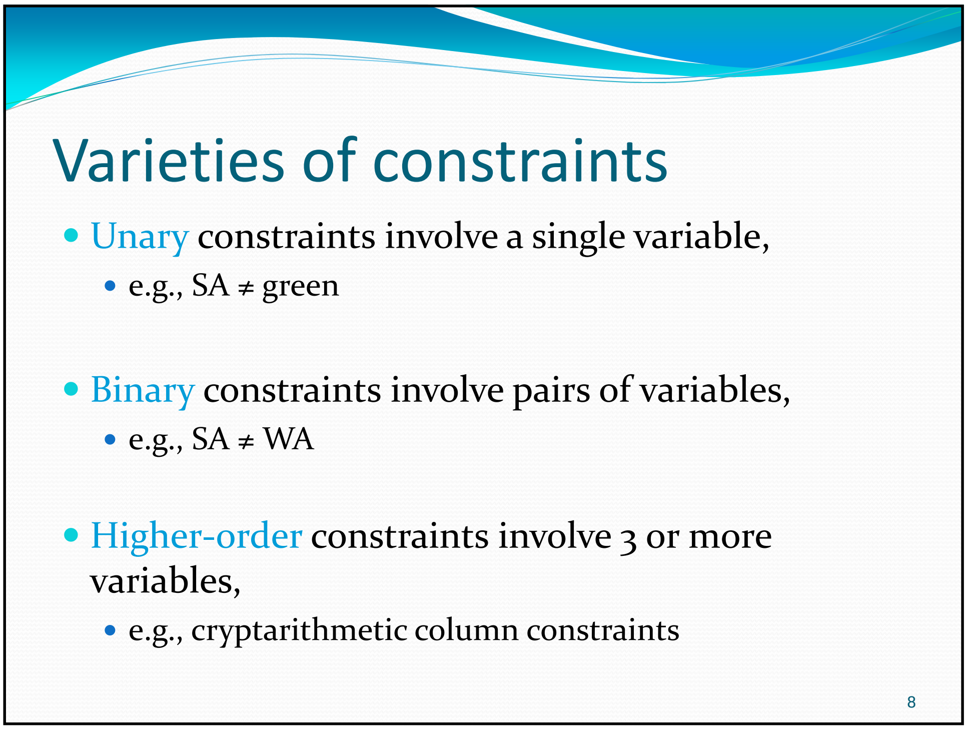
- Binary CSP: each constraint relates two variables
- Constraint graph: nodes are variables, arcs are constraints





Varieties of CSPs

- Discrete variables
  - finite domains:
    - *n* variables, domain size *d* → $O(d^n)$ complete assignments
    - e.g., Boolean CSPs, includes Boolean satisfiability (NP-complete)
  - infinite domains:
    - integers, strings, etc.
    - e.g., job scheduling, variables are start/end days for each job
    - need a constraint language, e.g., $StartJob_1 + 5 \leq StartJob_3$
- Continuous variables
  - e.g., start/end times for Hubble Space Telescope observations
  - linear constraints solvable in polynomial time by linear programming

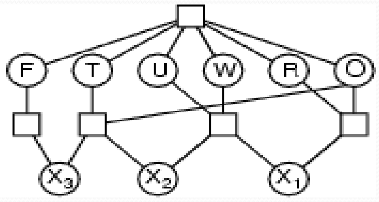


Varieties of constraints

- Unary constraints involve a single variable,
  - e.g., SA ≠ green
- Binary constraints involve pairs of variables,
  - e.g., SA ≠ WA
- Higher-order constraints involve 3 or more variables,
  - e.g., cryptarithmetic column constraints

Higher Order Constraint Example: Cryptarithmetic

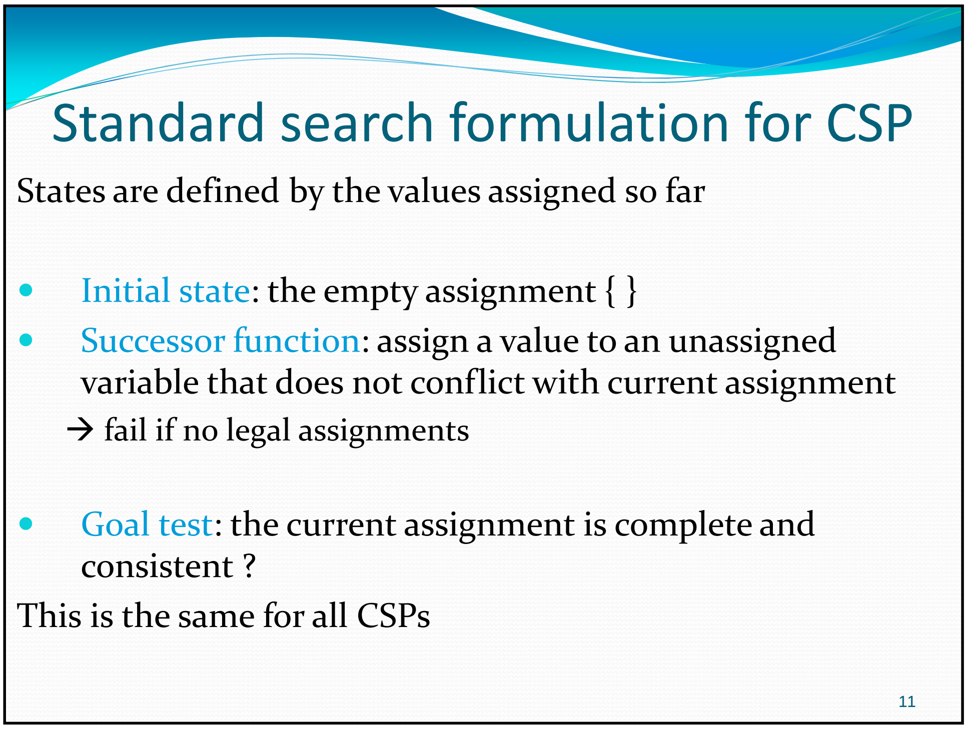
```
   T W O
 + T W O
 F O U R
```



- Variables: $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$
- Domains: $\{0,1,2,3,4,5,6,7,8,9\}$
- Constraints:
  - *alldifferent(F,T,U,W,R,O)*
  - $O + O = R + 10 \cdot X_1$
  - $X_1 + W + W = U + 10 \cdot X_2$
  - $X_2 + T + T = O + 10 \cdot X_3$
  - $X_3 = F,\ T \neq 0,\ F \neq 0$

Real-world CSPs

- Assignment problems
  - e.g., who teaches what class ?
- Timetabling problems
  - e.g., which class is offered when and where?
- Transportation scheduling
  - e.g., which car uses which road and when ?
- Factory scheduling
  - e.g., which job is performed when and about which part ?
- Notice that many real-world problems involve real-valued variables
  - e.g., “when” value can be a real number !



Standard search formulation for CSP

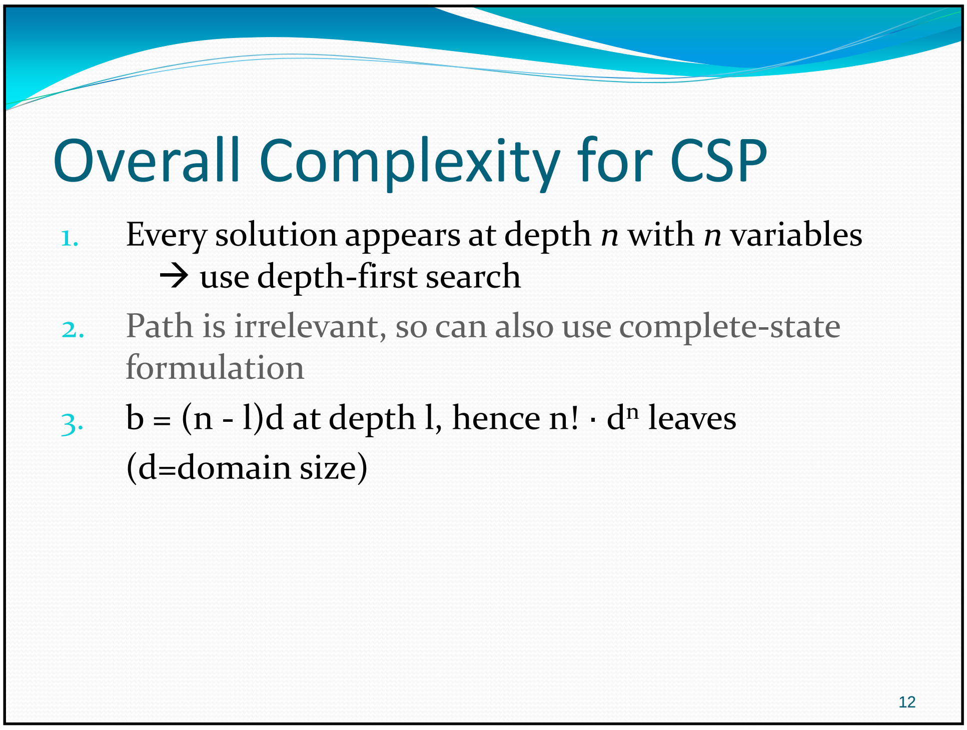
States are defined by the values assigned so far

- Initial state: the empty assignment { }
- Successor function: assign a value to an unassigned variable that does not conflict with current assignment

  → fail if no legal assignments

- Goal test: the current assignment is complete and consistent ?

This is the same for all CSPs



Overall Complexity for CSP

1. Every solution appears at depth *n* with *n* variables

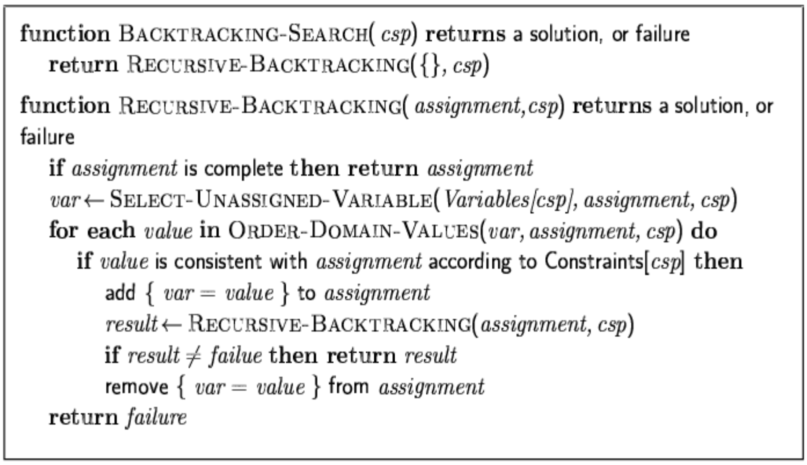
   → use depth-first search
2. Path is irrelevant, so can also use complete-state formulation
3. b = (n - l)d at depth l, hence n! · d^n leaves

   (d=domain size)

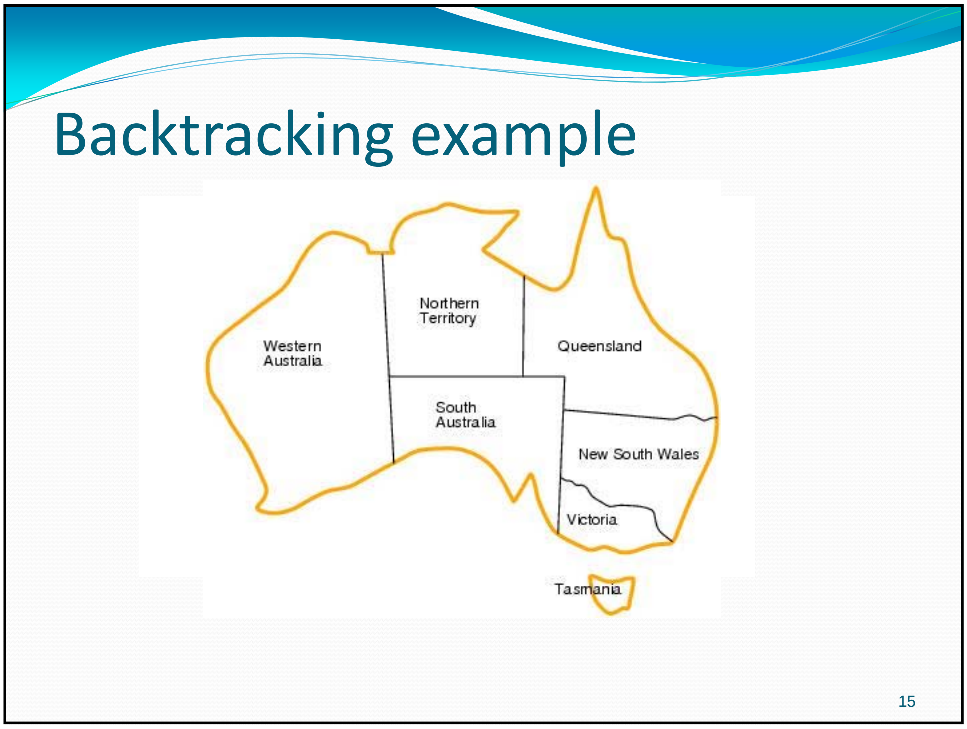
Backtracking search

- Variable assignments are commutative
  - i.e., “WA = red then NT = green” is the same as “NT = green then WA = red”
- Only need to consider assignments to a single variable at each node
  → b = d and there are d^n leaves
- Depth-first search for CSPs with single-variable assignments is called backtracking search
- Backtracking search is the basic uninformed algorithm for CSPs
- Can solve *n*-queens for $n \approx 25$

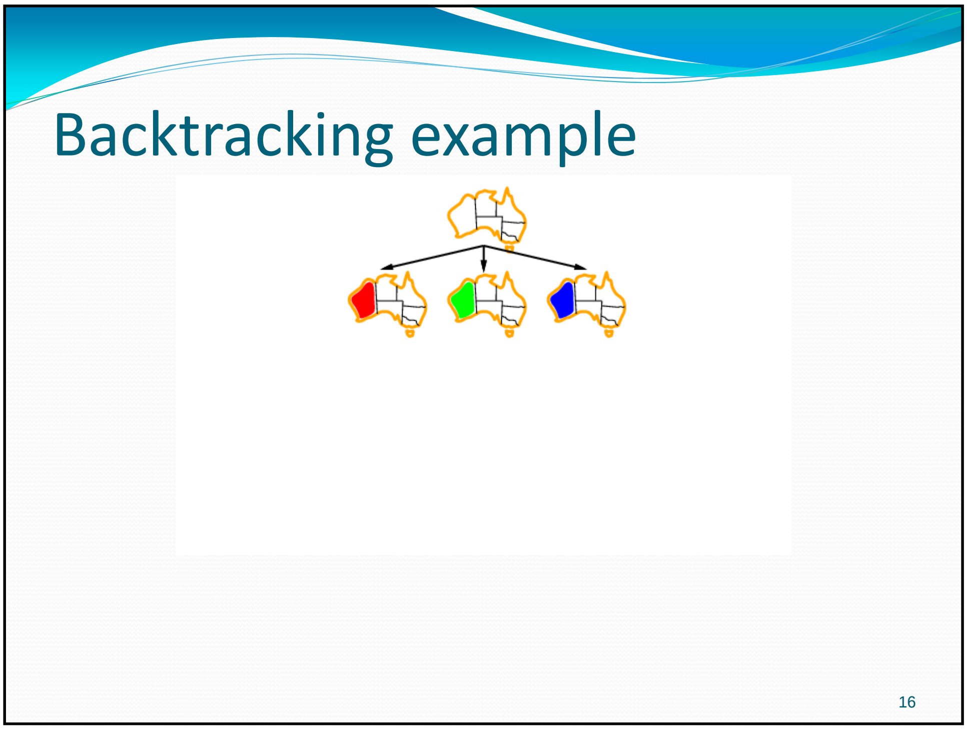
Backtracking search



```
function BACKTRACKING-SEARCH( csp) returns a solution, or failure
   return RECURSIVE-BACKTRACKING({}, csp)
function RECURSIVE-BACKTRACKING( assignment,csp) returns a solution, or
failure
   if assignment is complete then return assignment
   var ← SELECT-UNASSIGNED-VARIABLE( Variables[csp], assignment, csp)
   for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment according to Constraints[csp] then
         add { var = value } to assignment
         result ← RECURSIVE-BACKTRACKING(assignment, csp)
         if result ≠ failue then return result
         remove { var = value } from assignment
   return failure
```

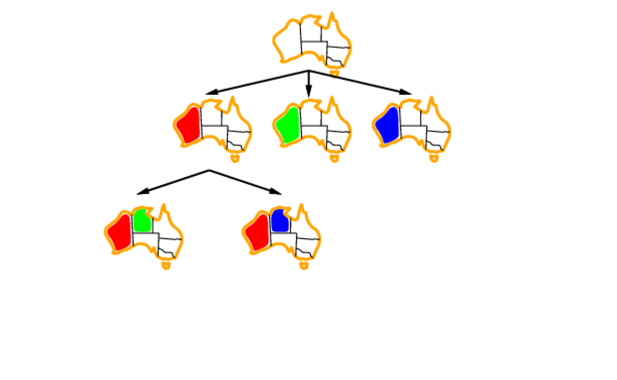


Backtracking example

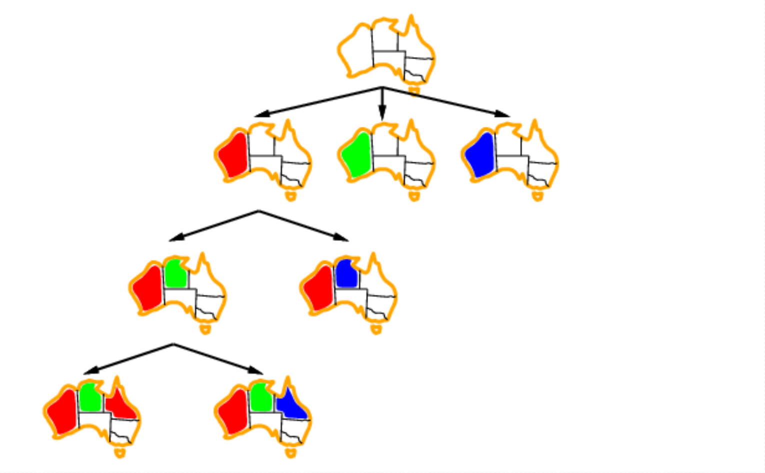


Backtracking example

Backtracking example



Backtracking example



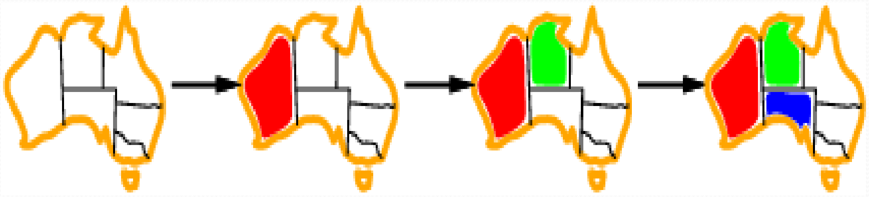
Improving backtracking efficiency

- Some general-purpose methods can give huge gains in speed:
  - Which variable should be assigned next?
  - In what order should its values be tried?
  - Can we detect inevitable failure early?
  - Can we take advantage of problem structure?

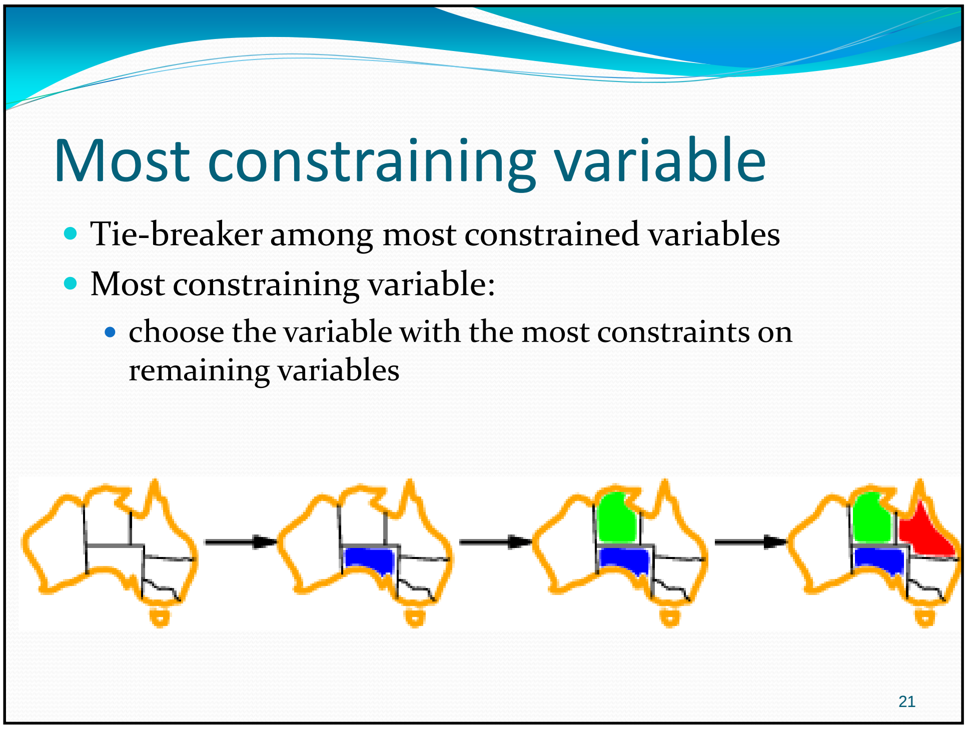
Most constrained variable

- Most constrained variable:

  choose the variable with the fewest legal values

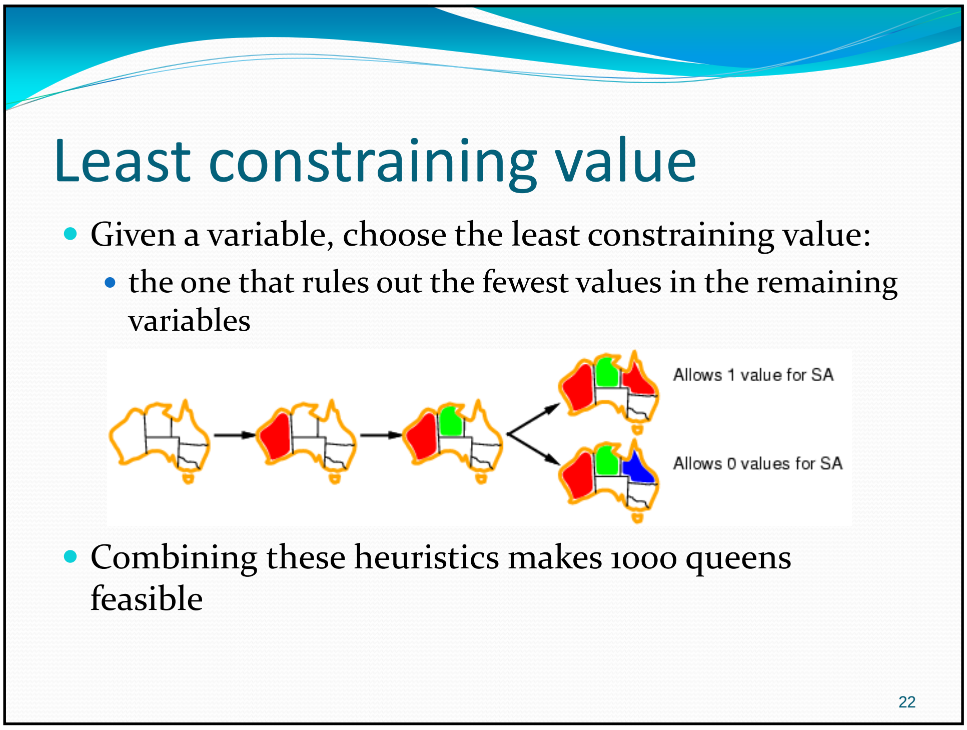


- minimum remaining values (MRV) heuristic



Most constraining variable

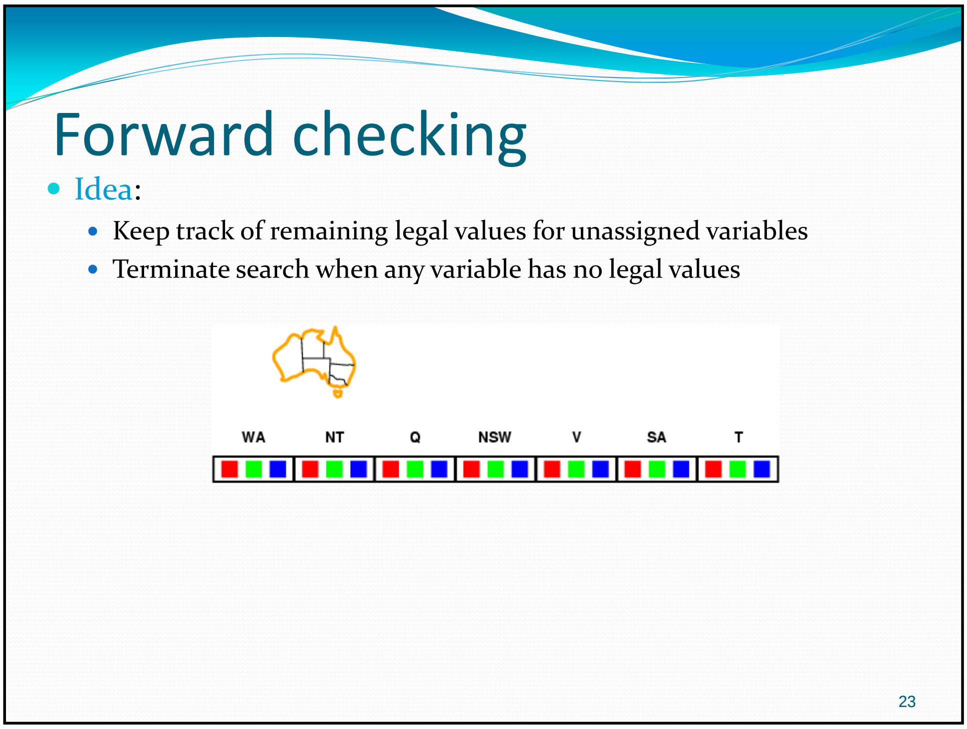
- Tie-breaker among most constrained variables
- Most constraining variable:
  - choose the variable with the most constraints on remaining variables



Least constraining value

- Given a variable, choose the least constraining value:
  - the one that rules out the fewest values in the remaining variables

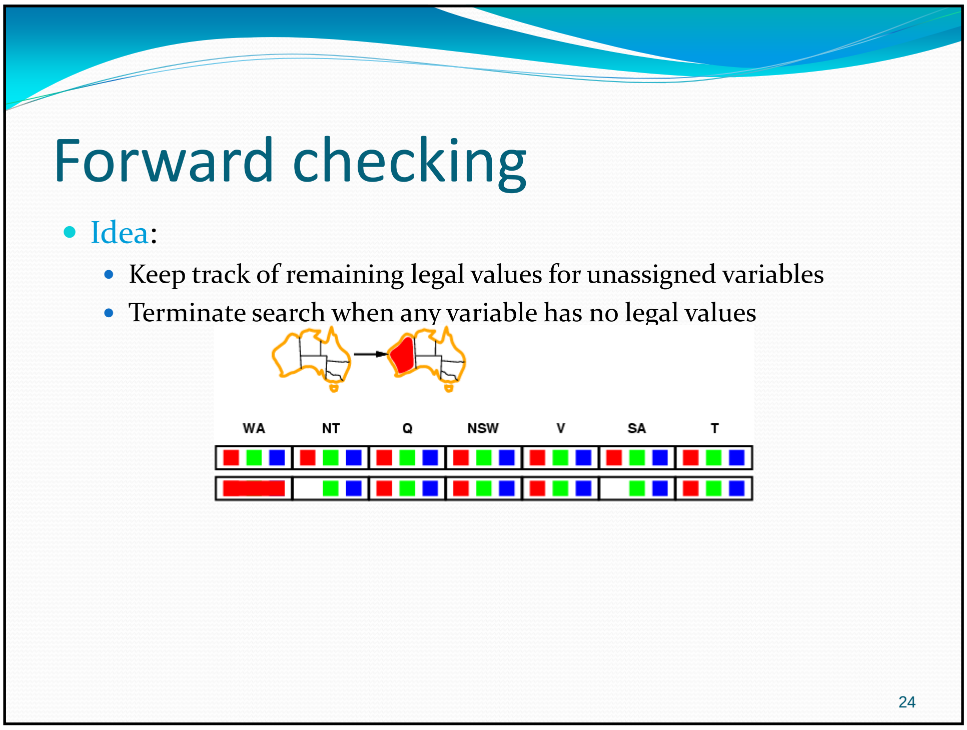
- Combining these heuristics makes 1000 queens feasible



Forward checking

- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values

WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦



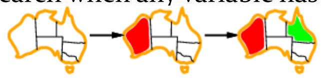
Forward checking

- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values

WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦

Forward checking

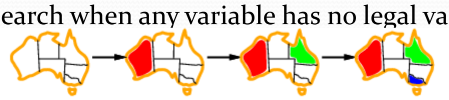
- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values



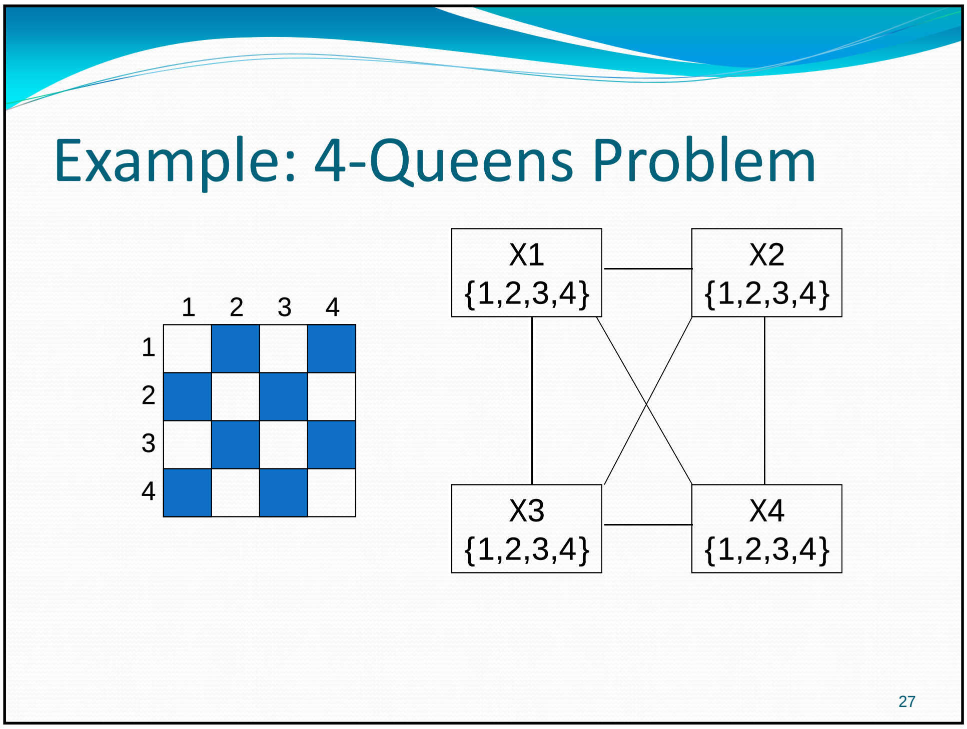
WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦

Forward checking

- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values



WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦
🟥	🟦	🟩	🟥	🟦		🟥🟩🟦



Example: 4-Queens Problem

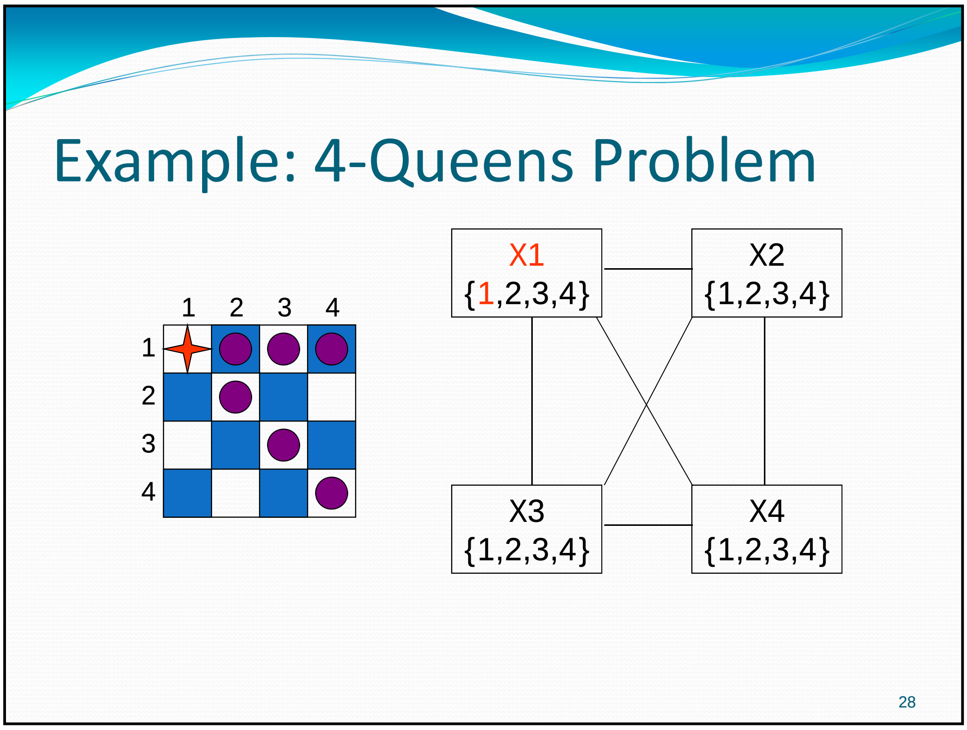
X1 {1,2,3,4}

X2 {1,2,3,4}

X3 {1,2,3,4}

X4 {1,2,3,4}

27



Example: 4-Queens Problem

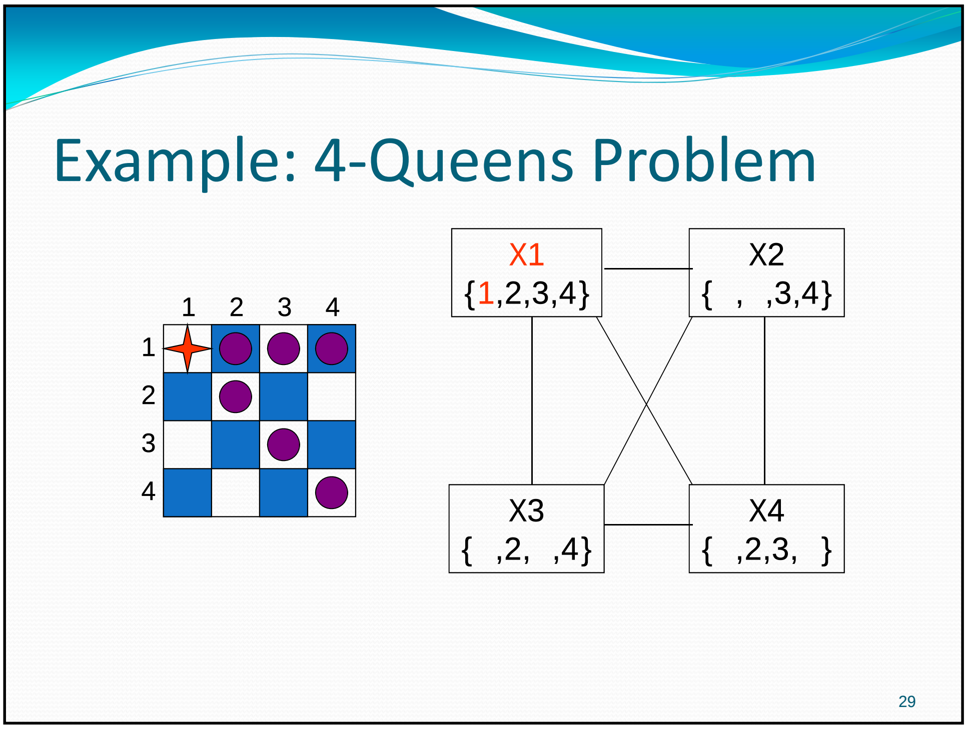
X1 {1,2,3,4}

X2 {1,2,3,4}

X3 {1,2,3,4}

X4 {1,2,3,4}

28



Example: 4-Queens Problem

	1	2	3	4
1	✦	●	●	●
2		●		
3			●	
4				●

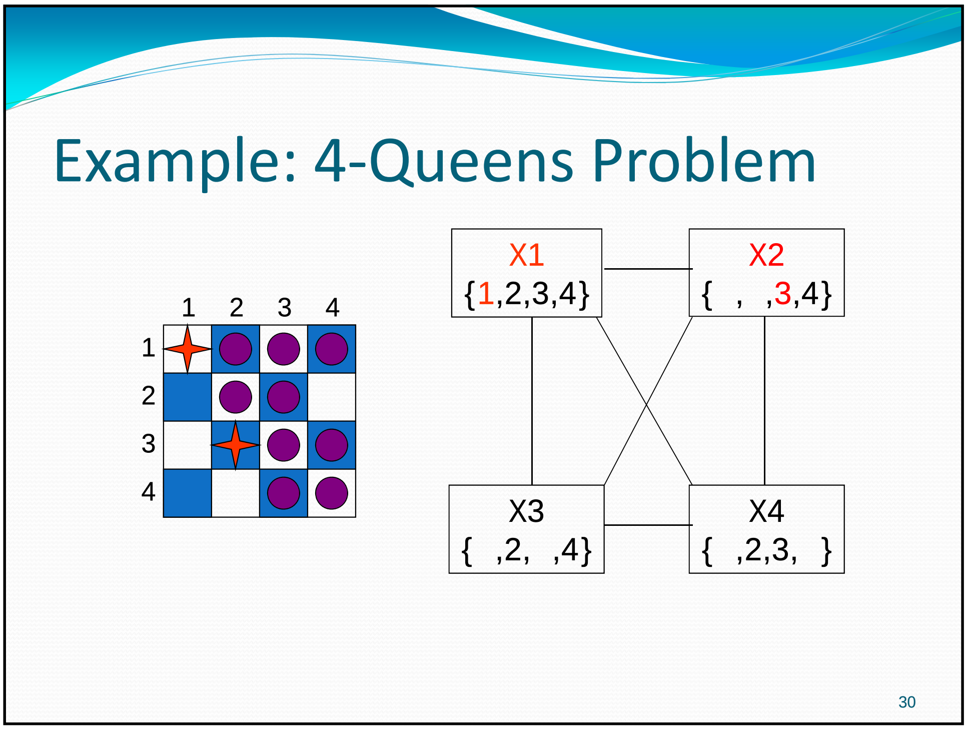
X1
{1,2,3,4}

X2
{ , ,3,4}

X3
{ ,2, ,4}

X4
{ ,2,3, }

29



Example: 4-Queens Problem

	1	2	3	4
1	✦	●	●	●
2		●	●	
3		✦	●	●
4			●	●

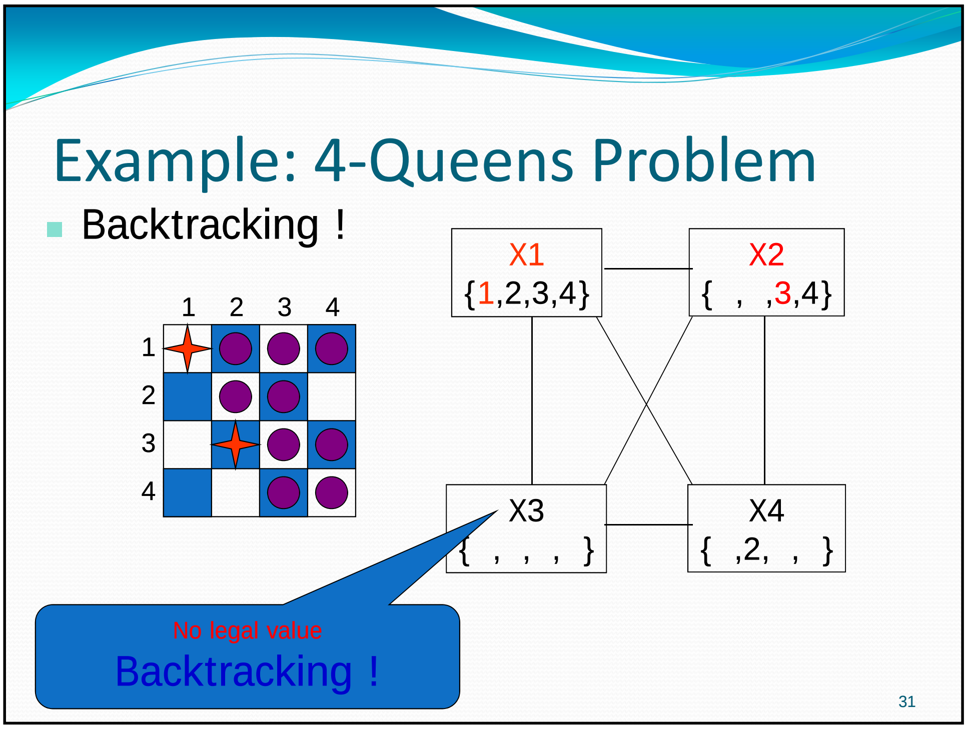
X1
{1,2,3,4}

X2
{ , ,3,4}

X3
{ ,2, ,4}

X4
{ ,2,3, }

30



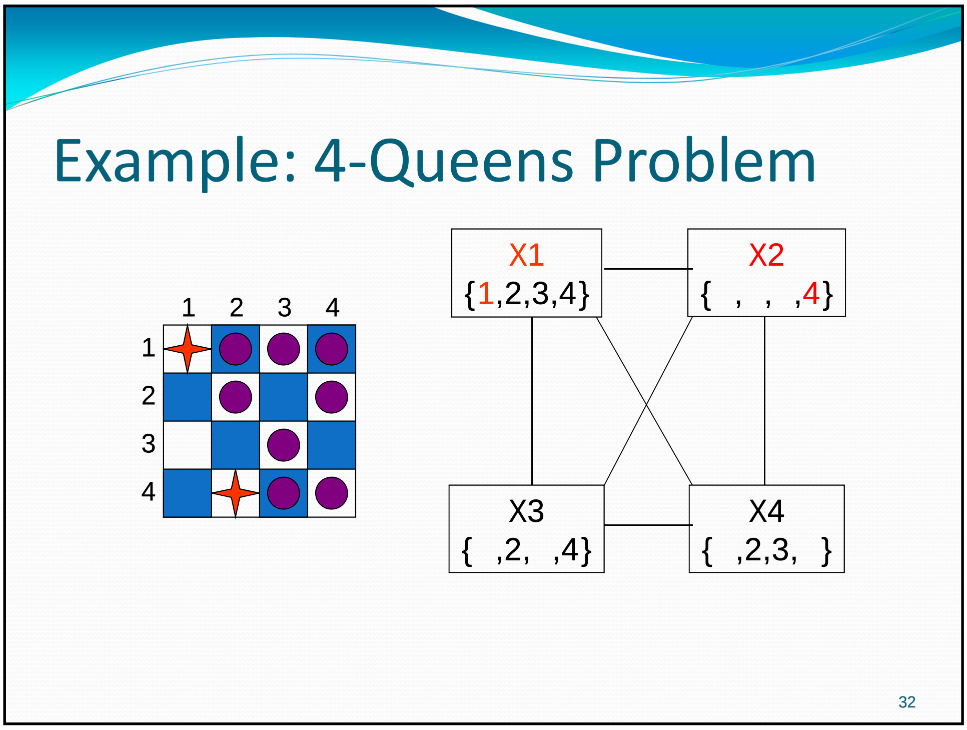
Example: 4-Queens Problem

- Backtracking !

	1	2	3	4
1	✦	●	●	●
2		●	●	
3		✦	●	●
4			●	●

No legal value

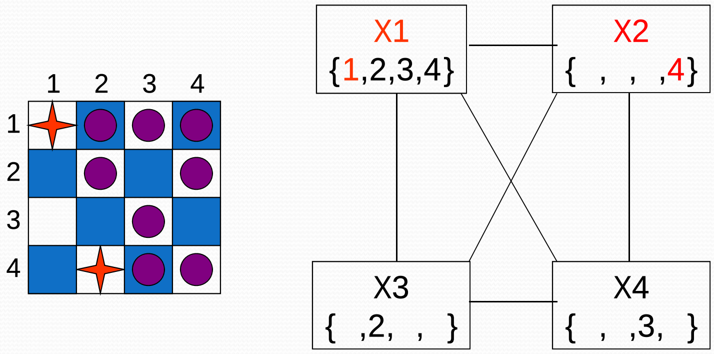
Backtracking !



Example: 4-Queens Problem

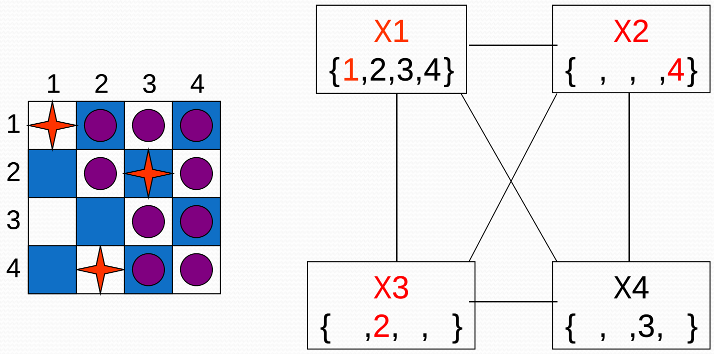
	1	2	3	4
1	✦	●	●	●
2		●		●
3			●	
4		✦	●	●

Example: 4-Queens Problem

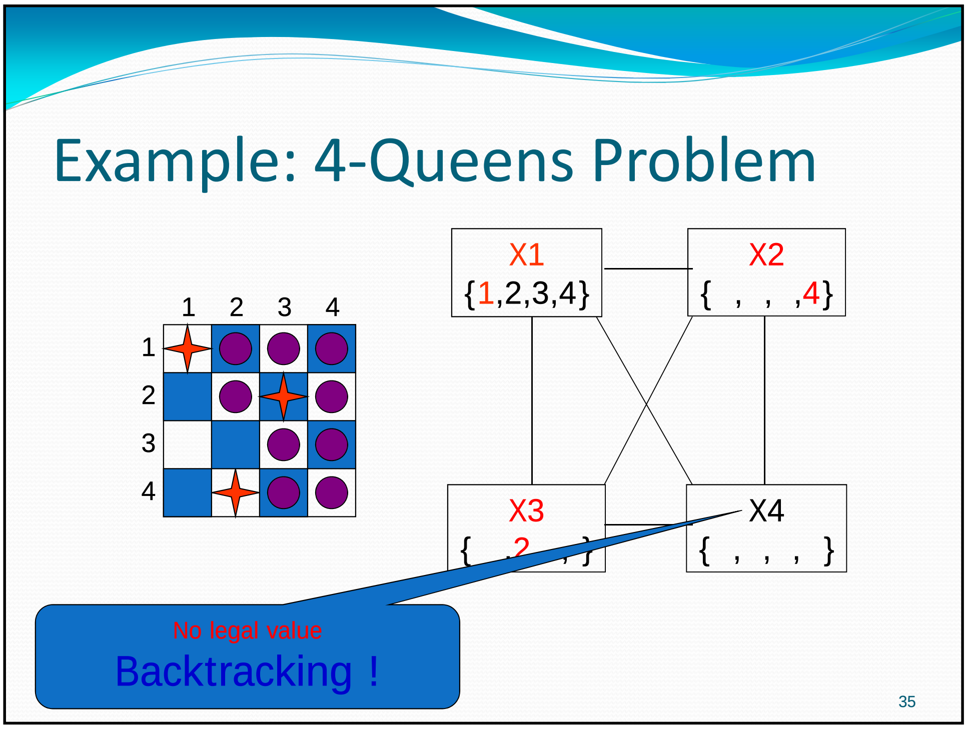


33

Example: 4-Queens Problem



34



Example: 4-Queens Problem

X1 {1,2,3,4}

X2 { , , ,4}

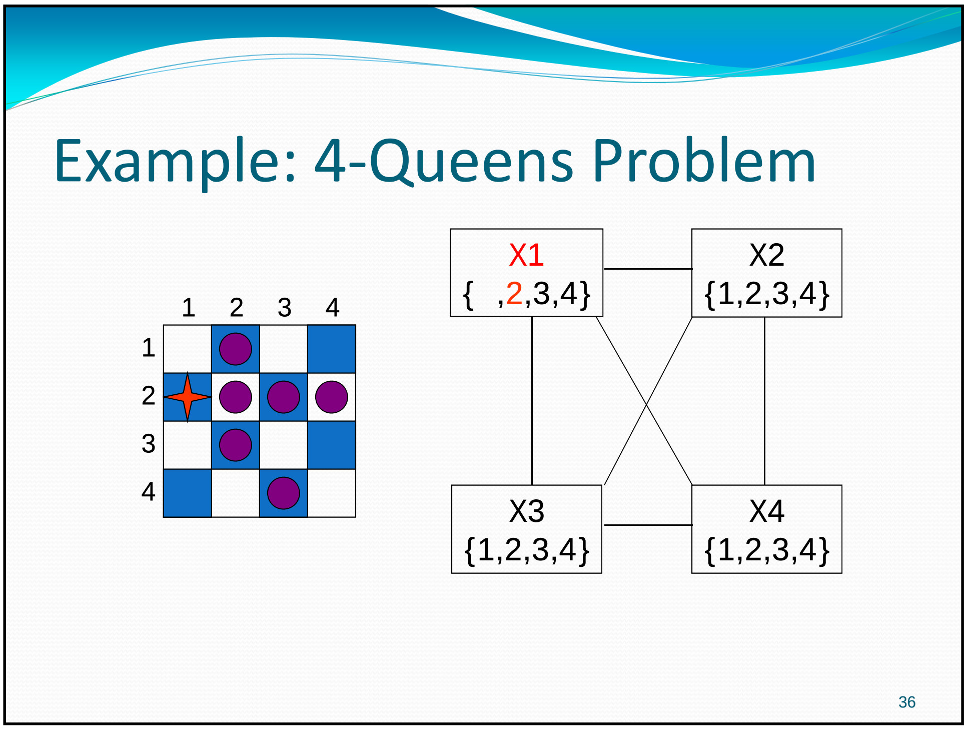
X3 { ,2, , }

X4 { , , , }

No legal value

Backtracking !

35



Example: 4-Queens Problem

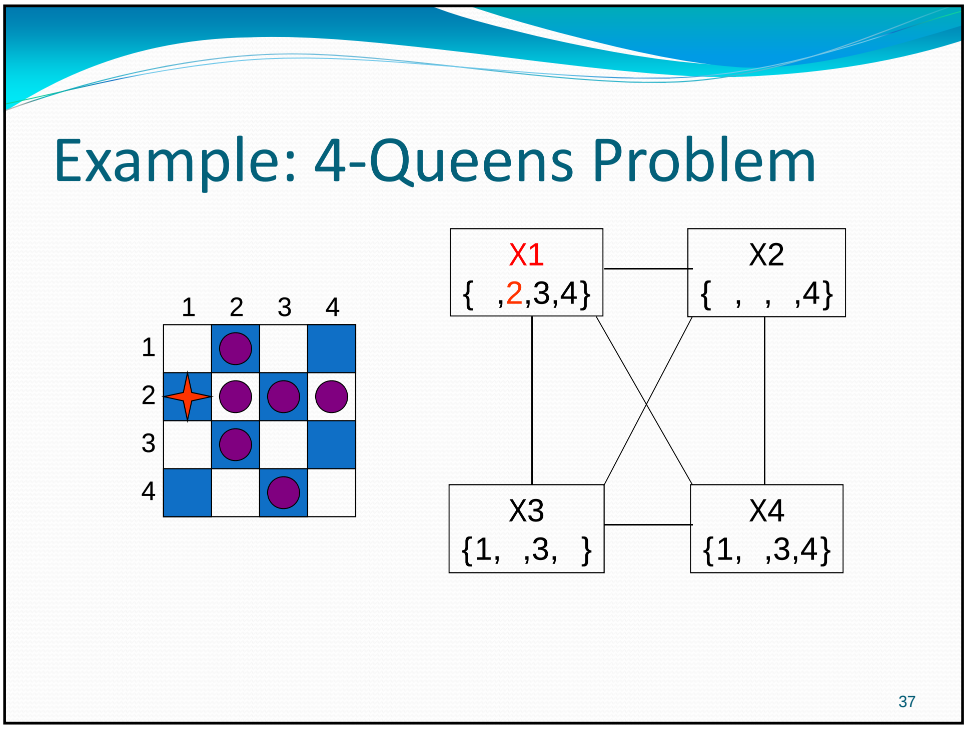
X1 { ,2,3,4}

X2 {1,2,3,4}

X3 {1,2,3,4}

X4 {1,2,3,4}

36



Example: 4-Queens Problem

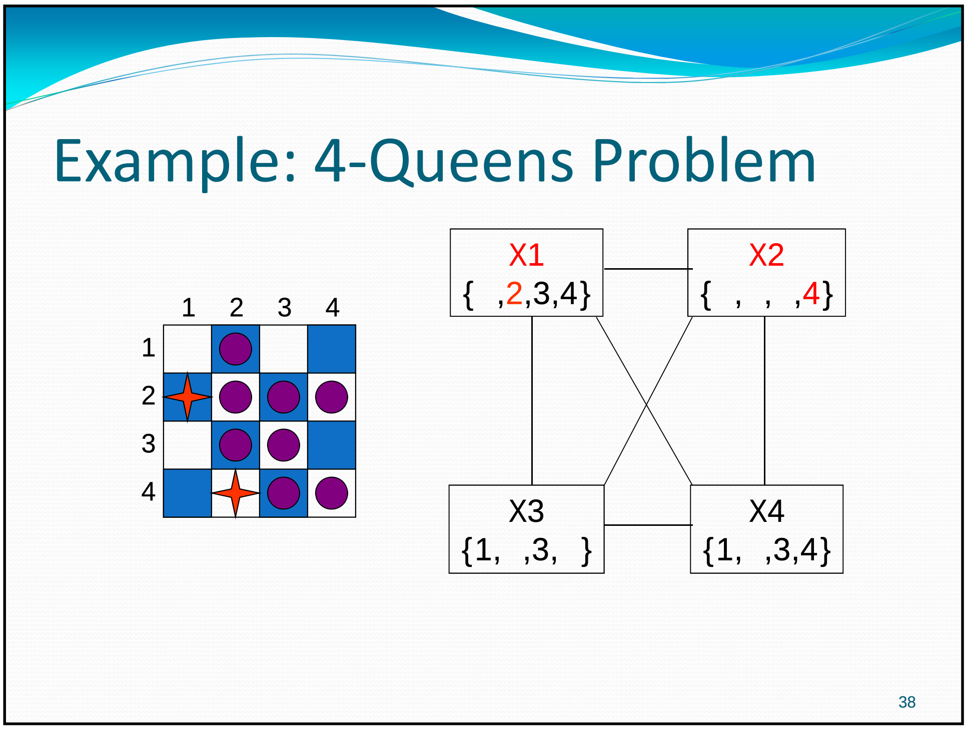
X1
{ ,2,3,4}

X2
{ , , ,4}

X3
{1, ,3, }

X4
{1, ,3,4}

37



Example: 4-Queens Problem

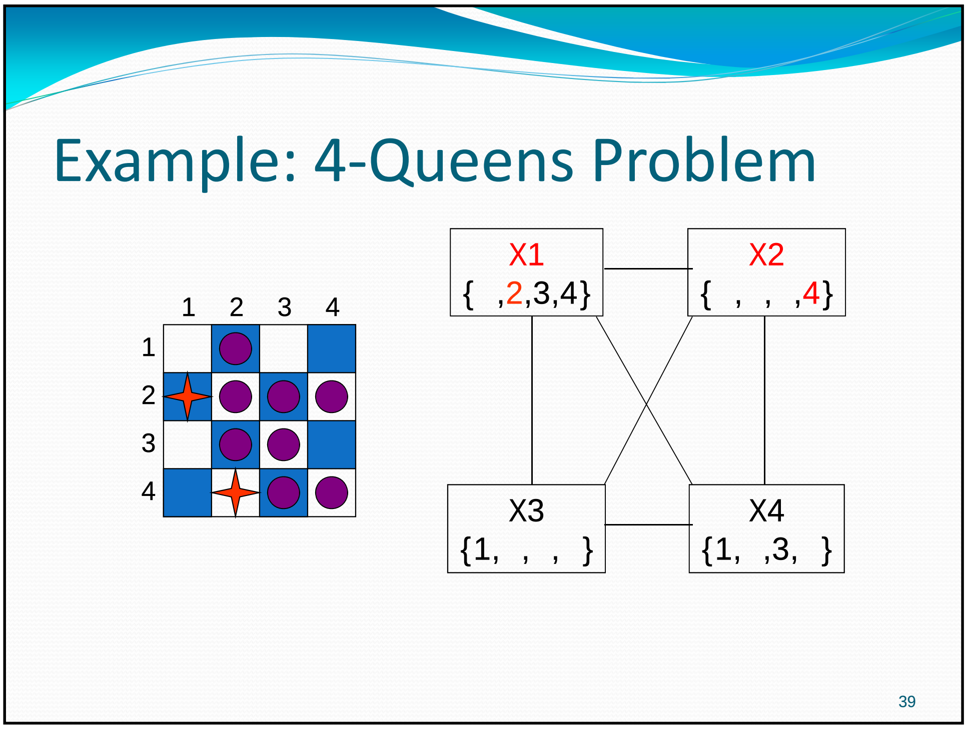
X1
{ ,2,3,4}

X2
{ , , ,4}

X3
{1, ,3, }

X4
{1, ,3,4}

38



Example: 4-Queens Problem

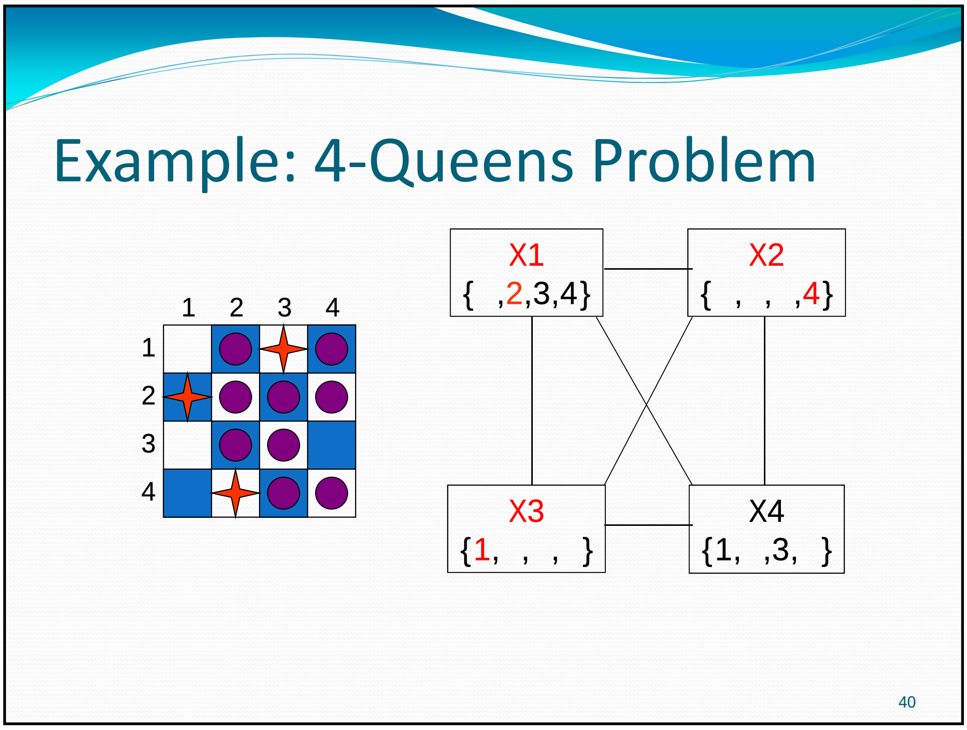
X1
{ ,2,3,4}

X2
{ , , ,4}

X3
{1, , , }

X4
{1, ,3, }

39



Example: 4-Queens Problem

X1
{ ,2,3,4}

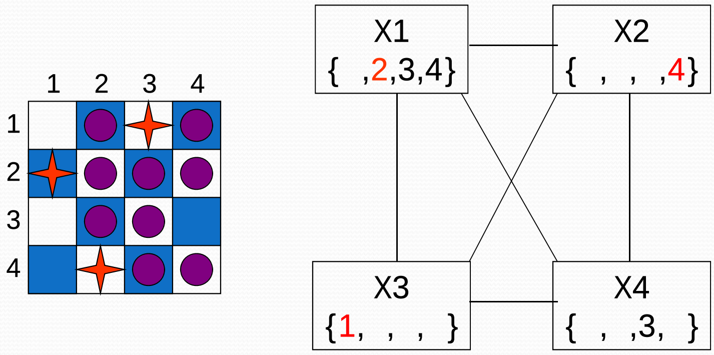
X2
{ , , ,4}

X3
{1, , , }

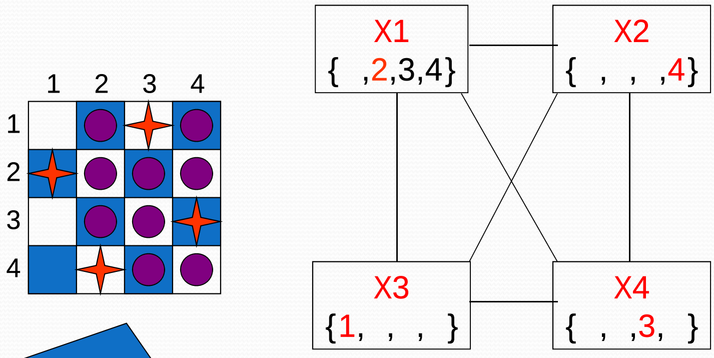
X4
{1, ,3, }

40

Example: 4-Queens Problem



Example: 4-Queens Problem



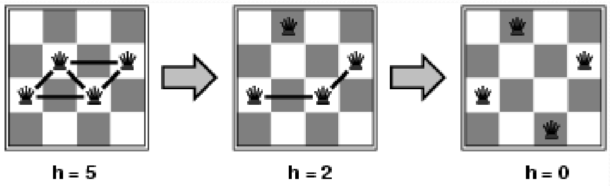
“X1=2, X2=4, X3=1, X4=3”
Solution Found !

Local search for CSPs

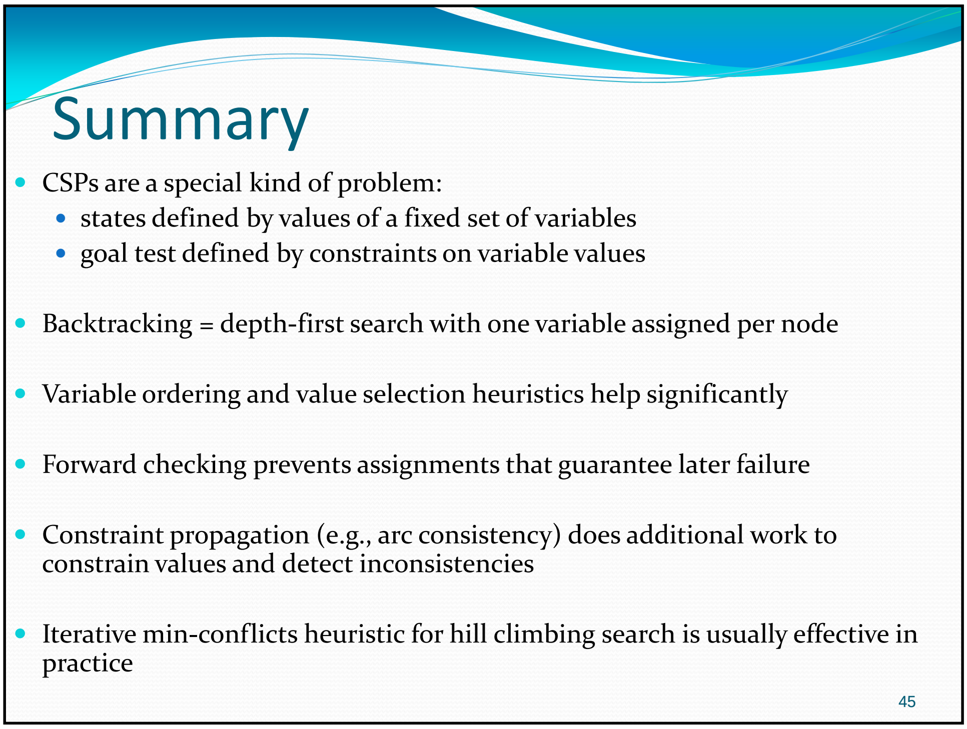
- Hill-climbing, simulated annealing typically work with "complete" states, i.e., all variables assigned
- To apply to CSPs:
  - allow states with unsatisfied constraints
  - operators reassign variable values
- Variable selection: randomly select any conflicted variable
- Value selection by min-conflicts heuristic:
  - choose value that violates the fewest constraints
  - i.e., hill-climb with *h(n)* = total number of violated constraints

Example: 4-Queens

- States: 4 queens in 4 columns (4^4 = 256 states)
- Actions: move queen in column
- Goal test: no attacks
- Evaluation: *h(n)* = number of attacks



- Given random initial state, can solve *n*-queens in almost constant time for arbitrary *n* with high probability (e.g., *n* = 10,000,000)



Summary

- CSPs are a special kind of problem:
  - states defined by values of a fixed set of variables
  - goal test defined by constraints on variable values
- Backtracking = depth-first search with one variable assigned per node
- Variable ordering and value selection heuristics help significantly
- Forward checking prevents assignments that guarantee later failure
- Constraint propagation (e.g., arc consistency) does additional work to constrain values and detect inconsistencies
- Iterative min-conflicts heuristic for hill climbing search is usually effective in practice