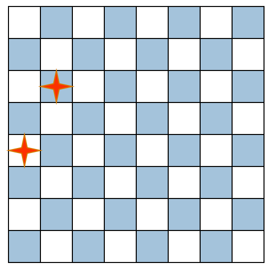
CONSTRAINT SATISFACTION

Today

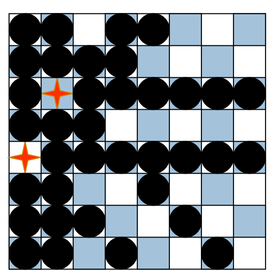
- Reading
  - AIMA Chapter 6
- Goals
  - Constraint satisfaction problems (CSPs)
  - Types of CSPs
  - Inference
  - Search + Inference

8-queens problem



How would you go about deciding where to put a third queen on the board in column 3?

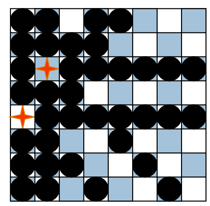
8-queens problem



How would you go about deciding where to put a third queen on the board in column 3?

8-queens problem

- This problem includes a set of constraints
- As a result, we need more than just a successor function and goal test
- We need a way to propagate the constraints imposed by one queen to the others and a way to detect early failure
  - Explicitly represent constraints
  - Algorithm to manipulate constraints

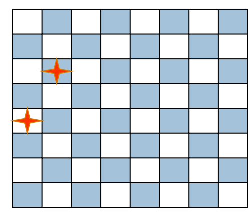


Constraint satisfaction problems

- Set of variables $\{X_1, X_2, \ldots, X_n\}$
- Each variable X_i has a domain D_i of possible values
- Set of constraints $\{C_1, C_2, \ldots, C_p\}$
  - Each constraint C_k involves a subset of variables and specifies the allowable combinations of values to these variables
- A state is an assignment of values to some or all of the variables
  - If the assignment doesn't violate any constraints we say it is consistent or legal
- The goal test is checking for a consistent and complete assignment

Example: 8-queens problems

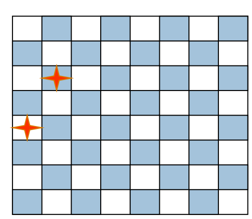
- Variable?
- Domain?
- Constraints?



Example: 8-queens problems

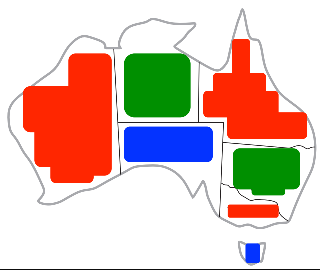
- Variables: one for each queen $\{X_1, \ldots, X_8\}$
- Domain: indicates row D = {1,2,…,8}
- Constraints:

$$X_i = k \Longrightarrow X_j \neq k \quad \forall i \neq j$$
$$X_i = k_i, X_j = k_j \Longrightarrow |i - j| \neq |k_i - k_j|$$



Example: Map coloring

- Variables: {WA, NT, SA, Q, NSW, V, T}
- Domains: {red, blue, green}
- Constraints: adjacent regions have different colors
  - Implicit: WA ≠ NT, WA ≠ SA, SA ≠ NT, NT ≠ Q,…
  - Explicit: (WA,NT) ∈ {(red,green), (red,blue),…}

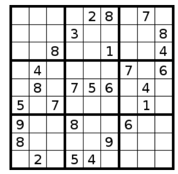


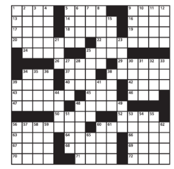
Example: Task scheduling

- Variables: {AxleF, AxleB, WheelRF, WheelLF,…,Inspect}
- Domains: Time task starts D = [0, 1, 2,…, ∞)
- Constraints:
  - Axle must be done before the wheel
    - AxleF + 10 < WheelLF
    - AxleF + 10 < WheelRF
  - The front axle and the back axle cannot be done at the same time
    - (AxleF + 10 < AxleB) OR (AxleB + 10 < AxleF)
  - Everything must be done within 30 minutes
    - Change domains to have upper bound 30 min.

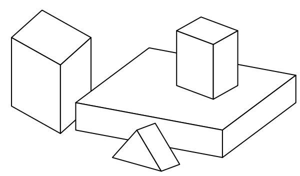
More examples

- More toy examples
  - sudoku, cryptarithmetic





- Real-world applications
  - Interpreting lines in 3D
  - Assignment problems, e.g. who teaches what class?
  - Timetable problems, e.g. which class offered when? where?
  - Transportation scheduling
  - Factory scheduling
  - Circuit layout



Types of CSPs - variables

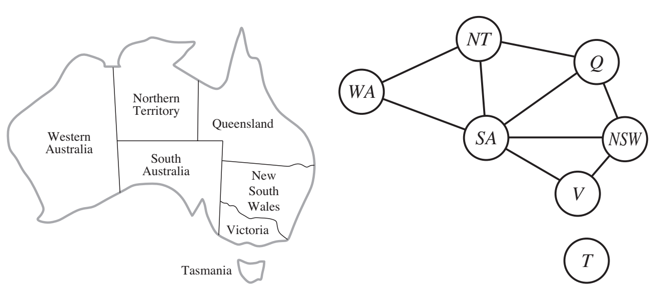
- Discrete variables
  - Finite domains
    - size d means $O(d^n)$ possible assignments to explore
  - Infinite domains
    - Linear constraints (e.g. $T_1 + d_1 \leq T_2$) are solvable
    - Non-linear constraints undecidable
- Continuous variables
  - linear programming problems with linear equality or inequality constraints solvable in polynomial time

Types of CSPs - constraints

- Unary constraints involve a single variable
  - e.g. SA ≠ green
- Binary constraints involve pairs of variables
  - SA ≠ NSW
  - A binary CSP can be illustrated using a constraint graph
- Higher-order constraints
  - e.g. A, B, and C cannot be in the same grouping
  - e.g. AllDiff (all variables must be assigned different values)
- Preference constraints
  - costs on individual variable assignments
  - constraint optimization problem

Constraint Graph

- Useful for binary constraint CSPs where each constraint relates (at most) two variables
- Nodes correspond to variables
- Edges (arcs) link two variables that participate in a constraint
- Use graph to speed up search



Inference: constraint propagation

- Use the constraints to reduce the number of legal values for a variable
- Possible to find a solution without searching
  - Node consistency
    - A node is node-consistent if all values in its domain satisfy the unary constraints
  - Arc consistency
    - A node X_i is arc-consistent w.r.t. node X_j if for every value in D_i there exists a value in D_j that satisfies the binary constraint
    - Algorithm AC-3
  - Other types of consistency (path consistency, k-consistency, global constraints)

AC-3 algorithm for Arc consistency

```
function AC-3(csp) returns false if inconsistency found, true otherwise
    queue ⟵ all arcs in csp
    while queue not empty
        (Xi,Xj) ⟵ REMOVE-FIRST(queue)
        if REMOVE-INCONSISTENT-VALUES(Xi, Xj)
            if size Di == 0 return false
            for each arc (Xk ,Xi)
                add (Xk, Xi) to queue
    return true
```

```
function REMOVE-INCONSISTENT-VALUES(Xi, Xj)
    revised ⟵ false
    for each x in Di
        if ∄ y in Dj s.t. (x,y) satisfies constraints
            delete x from Di
            revised ⟵ true
    return revised
```

AC-3 algorithm for Arc consistency

```
function AC-3(csp) returns false if inconsistency found, true otherwise
    queue ⟵ all arcs in csp
    while queue not empty
        (Xi,Xj) ⟵ REMOVE-FIRST(queue)
        if REMOVE-INCONSISTENT-VALUES(Xi, Xj)
            if size Di == 0 return false
            for each arc (Xk ,Xi)
                add (Xk, Xi) to queue
    return true
```

c constraints (arcs)
d domain size

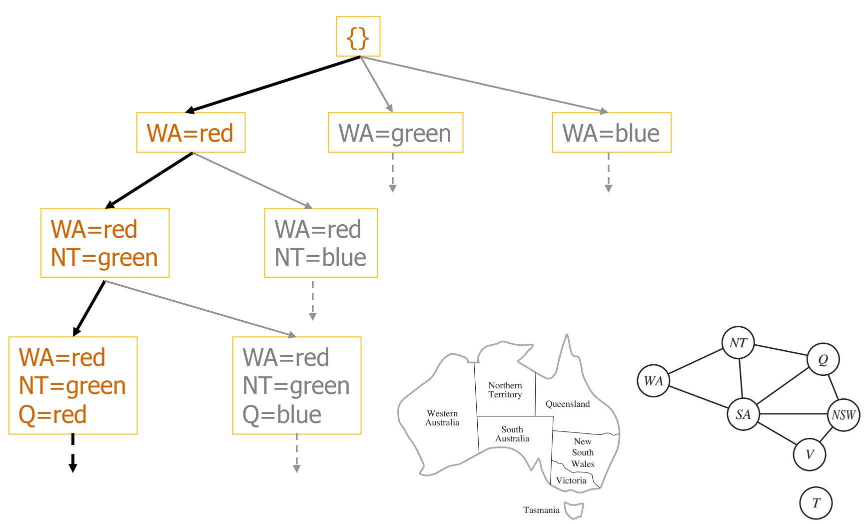
O(cd)

Total $O(cd^3)$

```
function REMOVE-INCONSISTENT-VALUES(Xi, Xj)
    revised ⟵ false
    for each x in Di
        if ∄ y in Dj s.t. (x,y) satisfies constraints
            delete x from Di
            revised ⟵ true
    return revised
```

$O(d^2)$

Search



Backtracking Search

```
function CSP-BACKTRACKING(assignment) returns a solution or failure
    if assignment complete return assignment
    X ⟵ select unassigned variable
    D ⟵ select an ordering for the domain of X
    for each value in D
        if value is consistent with assignment
            add (X = value) to assignment
            (ADD INFERENCE HERE)
            result ⟵ CSP-BACKTRACKING(assignment)
            if result ≠ failure return result
        remove (X = value) from assignment
    return failure
```

Backtracking Search

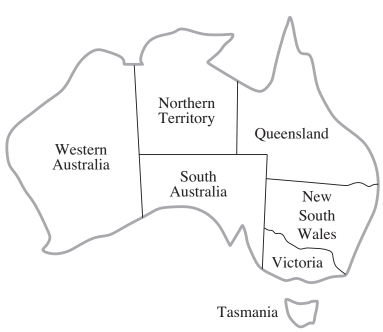
- Backtracking search is the basic uninformed algorithm for solving CSPs
- Idea 1: One variable at a time
  - Variable assignments are commutative so fix ordering
  - i.e. (WA = red, NT = green) is the same as (NT = green, WA = red)
- Idea 2: Check constraints as we go
  - Consider only values which do not conflict with previous assignments
  - May take some computation to check
  - "incremental goal test"
  - (Additional inference is optional, e.g. arc-consistency)
- Depth-first search with these 2 improvements is called *backtracking search*

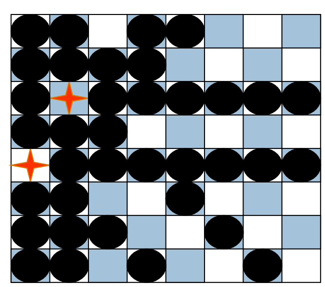
Improving backtracking search

- Ordering
  - Which variable X should be assigned a value next?
  - In which order should its domain D be sorted?
- Incorporating inference
  - Can we detect inevitable failure early?
- Structure
  - Can we exploit the problem structure?

Ordering

- Which variable should we choose?





Ordering

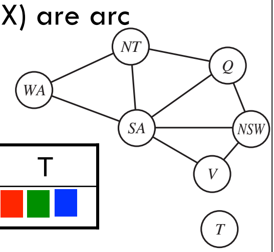
- Variable ordering
  - Minimum-remaining values heuristic - Choose the variable with the fewest "legal" moves remaining
  - Degree heuristic - Choose variable involved in the largest number of constraints with remaining unassigned variables
- Value ordering
  - Least-constraining value heuristic - Choose the value that rules out the fewest choices for the neighboring variables

Improving backtracking search

- Ordering
  - Which variable X should be assigned a value next?
  - In which order should its domain D be sorted?
- Incorporating inference
  - Can we detect inevitable failure early?
- Structure
  - Can we exploit the problem structure?

Incorporating inference: forward checking

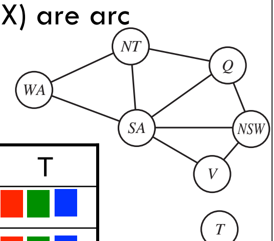
- After an assignment X = x, look at each unassigned variable Y connected to X by a constraint
  - Delete from Y's domain any value inconsistent with X = x
  - Equivalent to ensuring all arcs of the form (Y, X) are arc consistent



WA	NT	Q	NSW	V	SA	T
Red Green Blue	Red Green Blue	Red Green Blue	Red Green Blue	Red Green Blue	Red Green Blue	Red Green Blue

Incorporating inference: forward checking

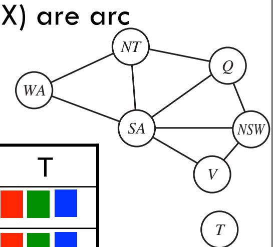
- After an assignment X = x, look at each unassigned variable Y connected to X by a constraint
  - Delete from Y's domain any value inconsistent with X = x
  - Equivalent to ensuring all arcs of the form (Y, X) are arc consistent

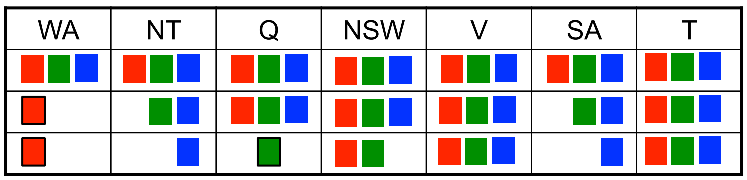


WA	NT	Q	NSW	V	SA	T
Red Green Blue	Red Green Blue	Red Green Blue	Red Green Blue	Red Green Blue	Red Green Blue	Red Green Blue
Red	Green Blue	Red Green Blue	Red Green Blue	Red Green Blue	Green Blue	Red Green Blue

Incorporating inference: forward checking

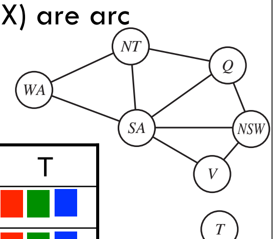
- After an assignment X = x, look at each unassigned variable Y connected to X by a constraint
  - Delete from Y's domain any value inconsistent with X = x
  - Equivalent to ensuring all arcs of the form (Y, X) are arc consistent

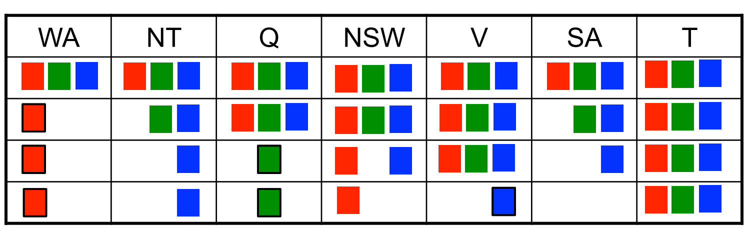




Incorporating inference: forward checking

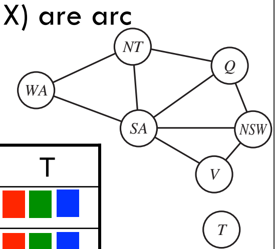
- After an assignment X = x, look at each unassigned variable Y connected to X by a constraint
  - Delete from Y's domain any value inconsistent with X = x
  - Equivalent to ensuring all arcs of the form (Y, X) are arc consistent

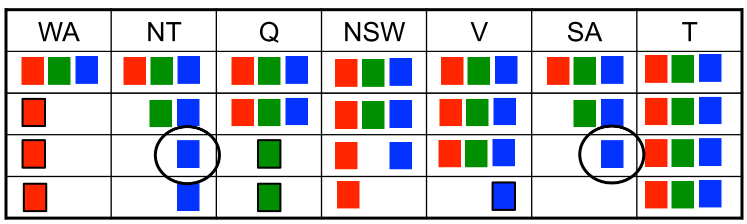




Incorporating inference: forward checking

- After an assignment X = x, look at each unassigned variable Y connected to X by a constraint
  - Delete from Y's domain any value inconsistent with X = x
  - Equivalent to ensuring all arcs of the form (Y, X) are arc consistent

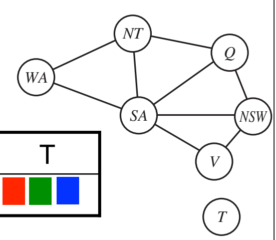


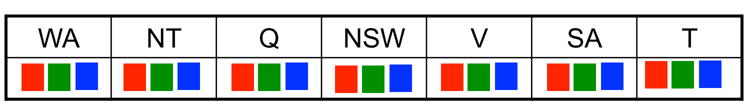


Could have detected earlier that things were going wrong!

Incorporating inference: AC-3

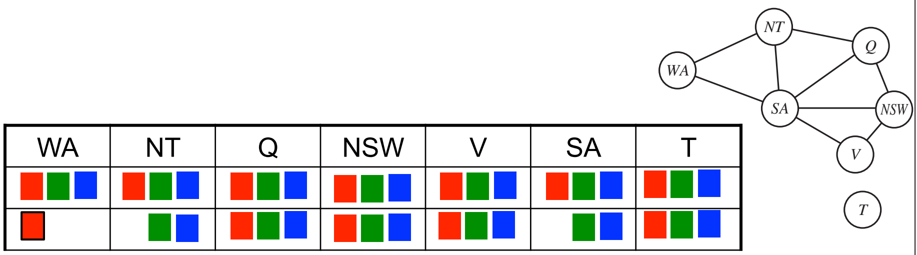
- Can also infer path consistency based on triples of variables and k-consistency





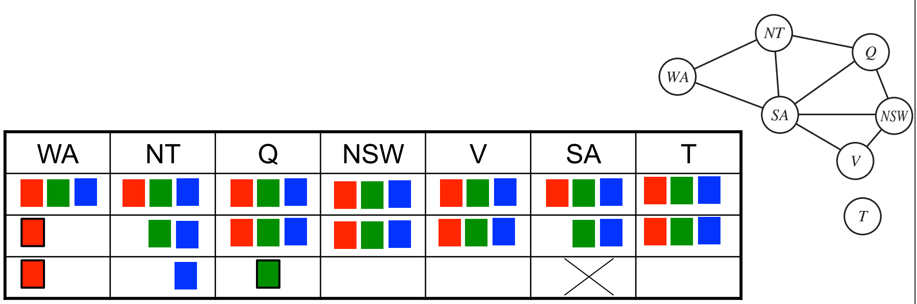
Incorporating inference: AC-3

- Can also infer path consistency based on triples of variables and k-consistency



Incorporating inference: AC-3

- Can also infer path consistency based on triples of variables and k-consistency

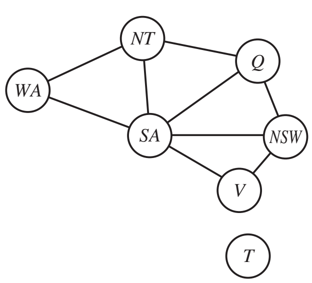


Improving backtracking search

- Ordering
  - Which variable X should be assigned a value next?
  - In which order should its domain D be sorted?
- Incorporating inference
  - Can we detect inevitable failure early?
- Structure
  - Can we exploit the problem structure?

Structure of the constraint graph

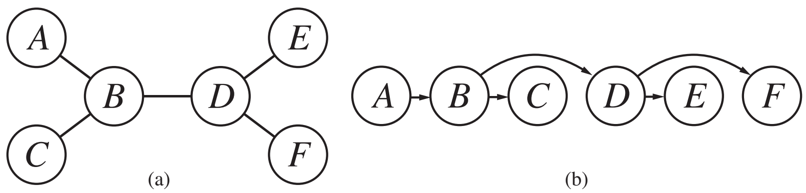
- Independent subproblems
  - Find connected components of the constraint graph
  - e.g. Tasmania and the mainland are independent
  - If we can split n variables into c subproblems of n/c variables each: $O(d^n) \longrightarrow O(d^c\, n/c)$



Tasmania is independent of the mainland!

Structure of the constraint graph

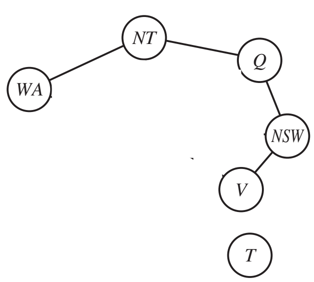
- Tree structured constraint graphs
  - Can solve in linear time using AC-3

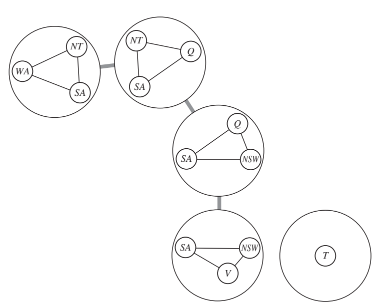


(a) (b)

Structure of the constraint graph

- Reduction to a tree structured graph
  - Cycle cutset – a subset of the variables whose removal creates a tree.
  - Tree decomposition – Divide graph into subproblems, solve independently merge the solutions

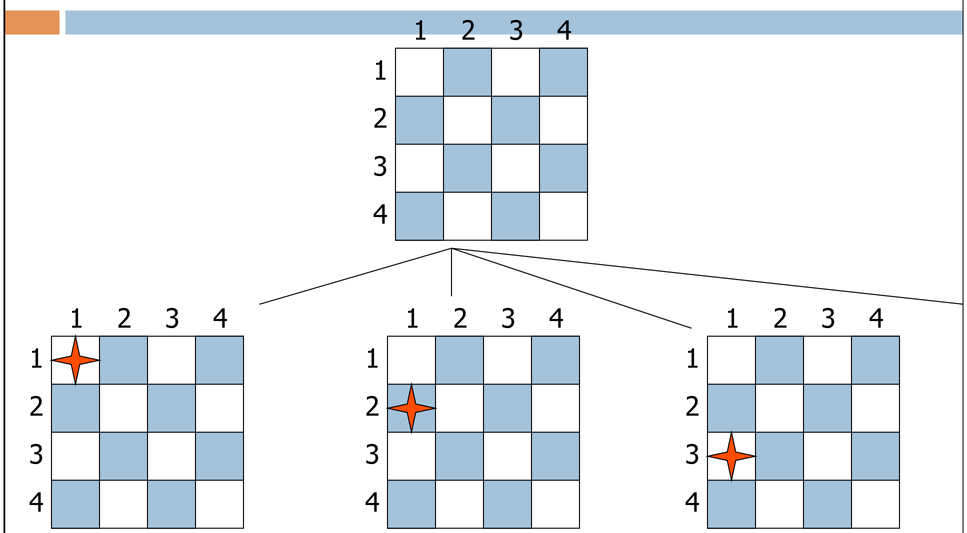




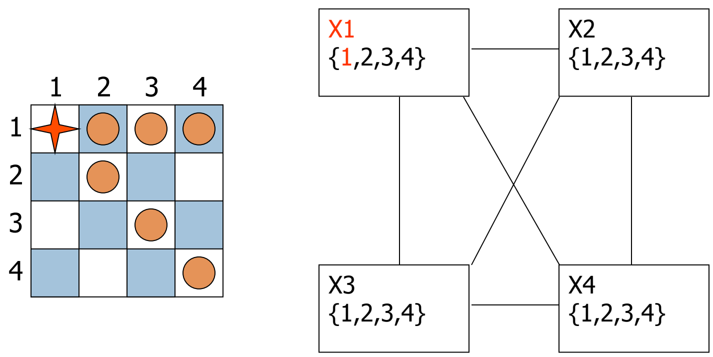
CSP Summary

- Constraint Satisfaction Problems (CSPs)
- Solving CSPs using inference
- Solving CSPs using search
  - Backtracking algorithm = DSF + constraints checking
  - General (not problem-specific) heuristics
- Improving Backtracking
  - Variable and value ordering
  - Inference
  - Structure

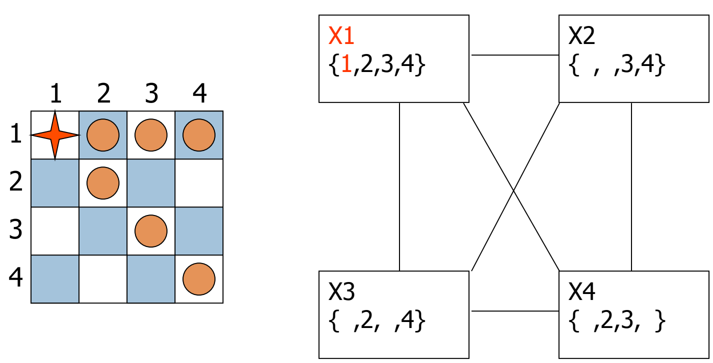
4-Queens Problem



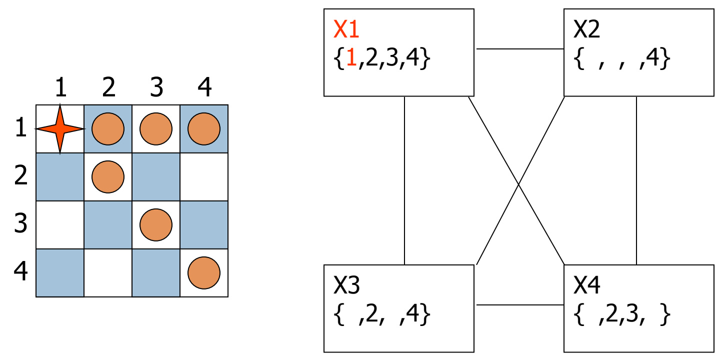
4-Queens Problem



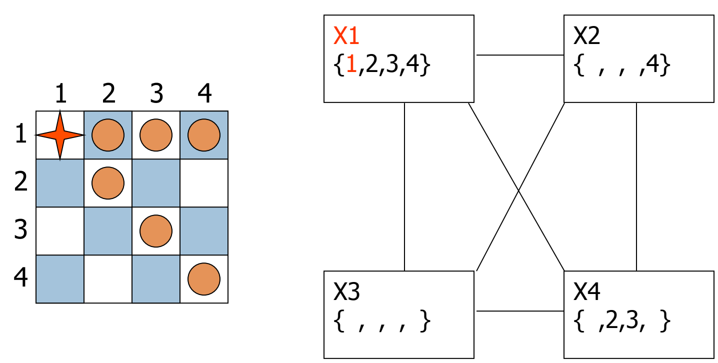
4-Queens Problem



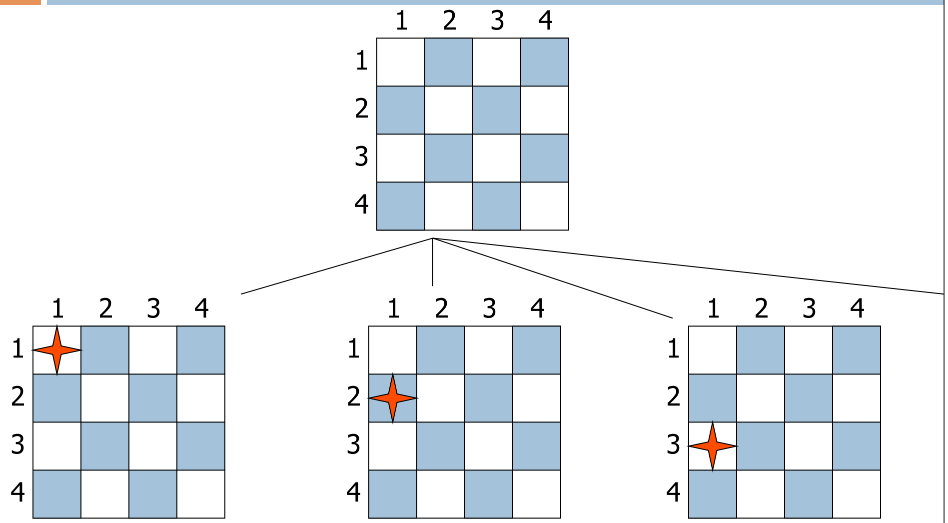
4-Queens Problem



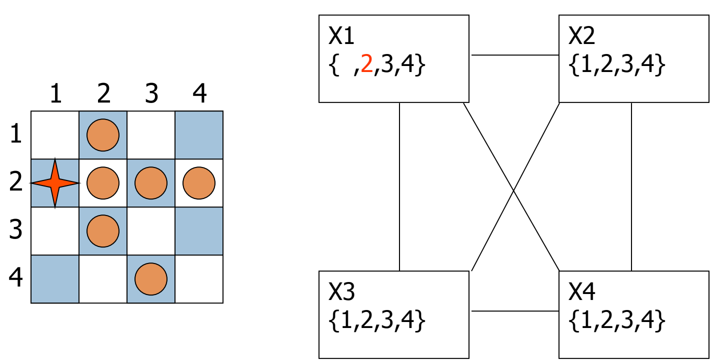
4-Queens Problem



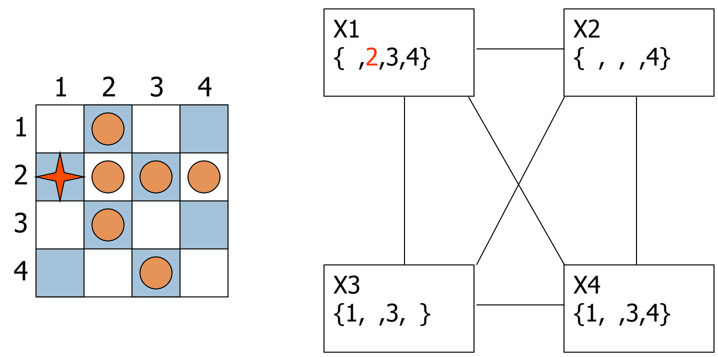
4-Queens Problem



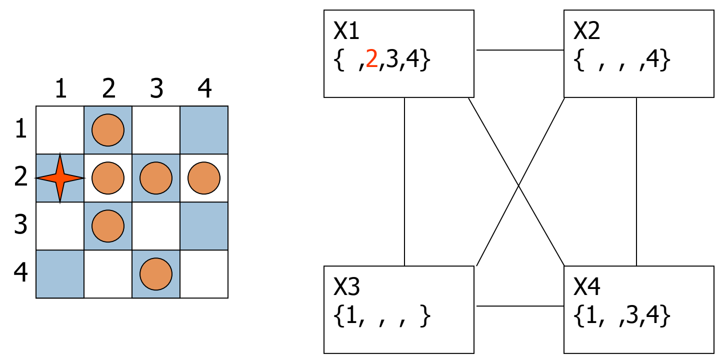
4-Queens Problem



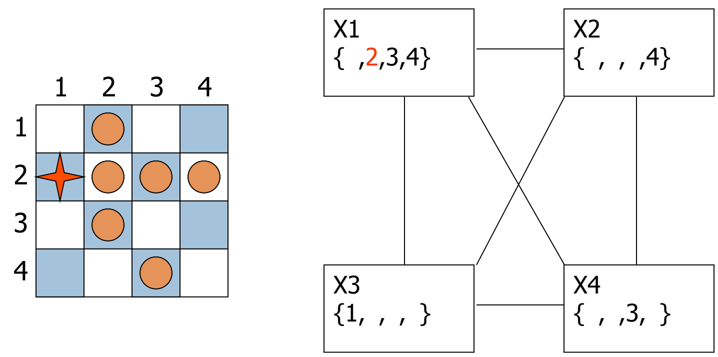
4-Queens Problem



4-Queens Problem



4-Queens Problem



4-Queens Problem