CS 480 – Introduction to Artificial Intelligence

Topic: Constraint Satisfaction
Chapter: 6

Mustafa Bilgic

http://www.cs.iit.edu/~mbilgic

https://twitter.com/bilgicm

Chapter 6 – Motivation

- Each state is described as a set of variables and their values
- There are constraints on which values can be assigned to which variables
- Examples
  - **Class scheduling:** Each room can host one class at a time, each instructor can be at one place at the same time, etc.
  - **Map coloring:** Adjacent states cannot have the same color
  - **Sudoku:** Numbers 1-9 must appear exactly once in a row, column, and block
- These are called Constraint Satisfaction Problems (CSP)
- We would like to develop general purpose and efficient solvers

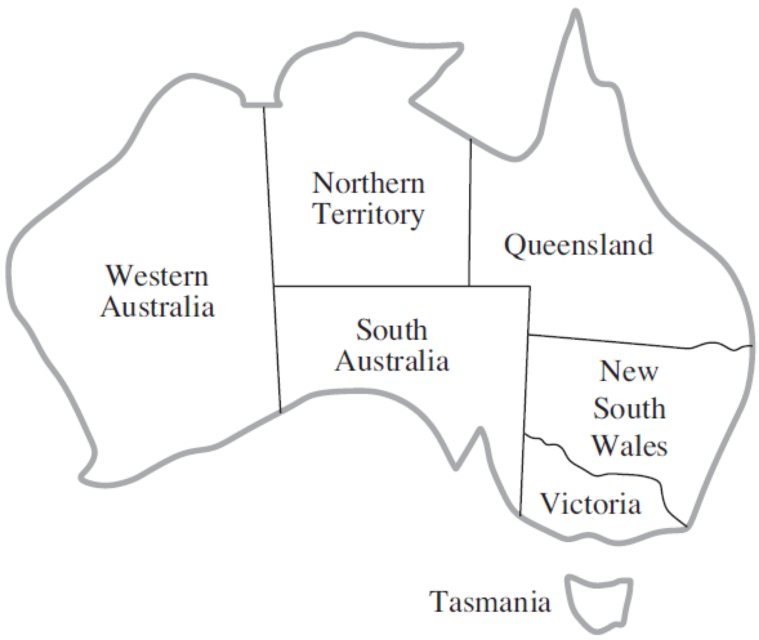
Outline

- Problem definition
- Constraint propagation
  - Node consistency
  - Arc consistency
  - Path consistency
  - K-consistency
  - Global constraints
- Backtracking search
  - Variable ordering
  - Value ordering
  - Forward checking
  - Maintaining arc consistency
- Local search

Problem Definition - CSP

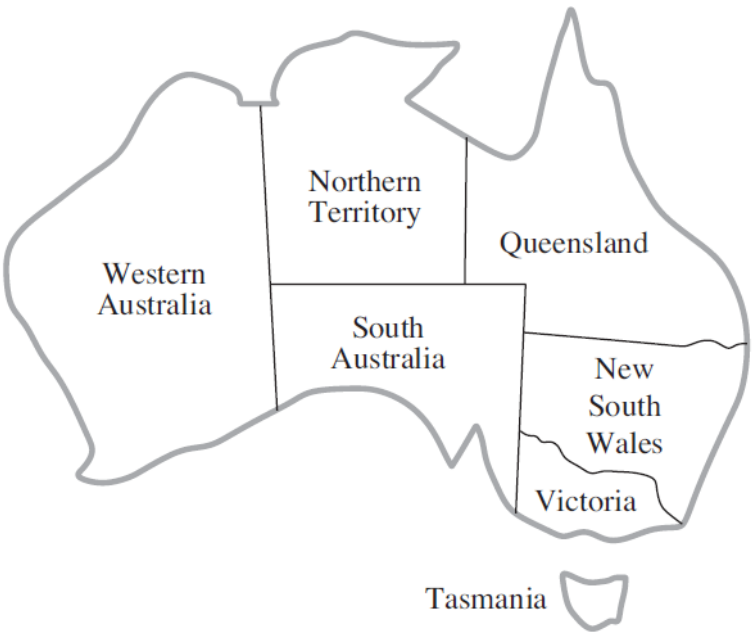
- $\mathcal{X}$ is a set of variables: $\{X_1, X_2, \ldots, X_n\}$
- $\mathcal{D}$ is a set of domains: $\{D_1, D_2, \ldots, D_n\}$, one for each variable
- $\mathcal{C}$ is a set of constraints on allowable combinations of values
- Definitions
  - **Assignment**: Some or all variables are assigned a value
  - **Consistent assignment**: No constraint is violated
  - **Complete assignment**: All variables are assigned
  - **Solution:** A consistent and complete assignment

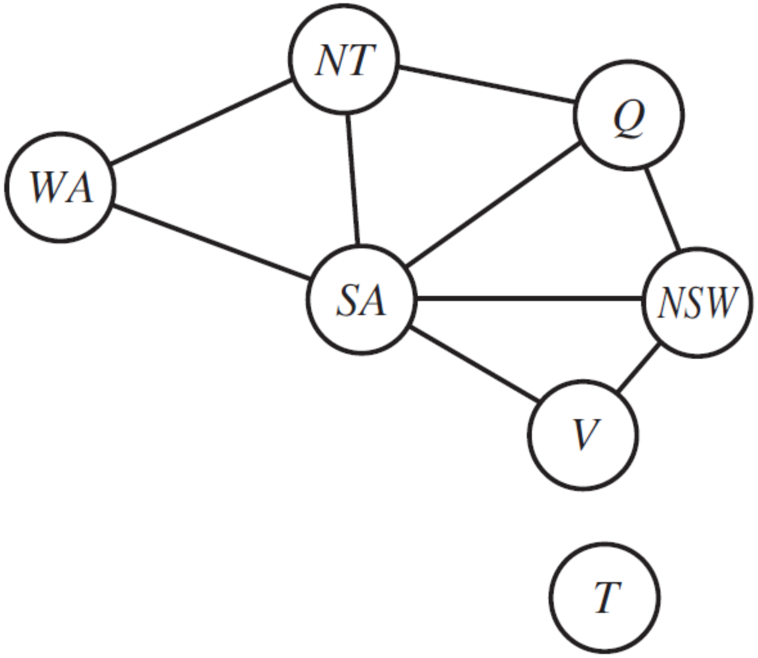
Example – Map Coloring



- X={WA, NT, SA, Q, NSW, V, T}
- D_i={red, $green$, $blue$}
- C={$WA \neq NT$, $WA \neq SA$, $NT \neq SA$, $NT \neq Q$, $SA \neq Q$, $SA \neq NSW$, $SA \neq V$, $Q \neq NSW$, $NSW \neq V$}

Constraint Graph





Variables are nodes. Two nodes are linked if they participate in a constraint.

Types of Constraints

- **Unary**: A variable cannot take on a value
- **Binary**: Constrains two variables
- **Global**: Constrains arbitrary number of variables
- We will consider binary CSPs
  - Unary: Modify the domain and remove the constraint
  - Global: Convert to binary constraint by introducing auxiliary variables

Solving CSPs

- A combination of
  - Search
    - Search for a value for a variable from its domain
  - Inference
    - Propagate constraints, reducing the domains of variables

Inference: Constraint Propagation

- **Node consistency**
  - Unary constraints are enforced by altering the domains
- **Arc consistency**
  - X_i is arc-consistent with respect to another variable X_j if for every value in the domain of D_i, there is some value in the domain of D_j that satisfies the binary constraint on the arc (X_i, X_j)

AC-3 ALGORITHM

- Put all arcs in a queue
- while queue is not empty
  - pick an arc (X_i, X_j) from the queue
  - make X_i arc consistent
  - if D_i is modified
    - if D_i is empty
      - return failure
    - else
      - Add all (X_k, X_i) to the queue, where X_k is a neighbor of X_i

AC-3

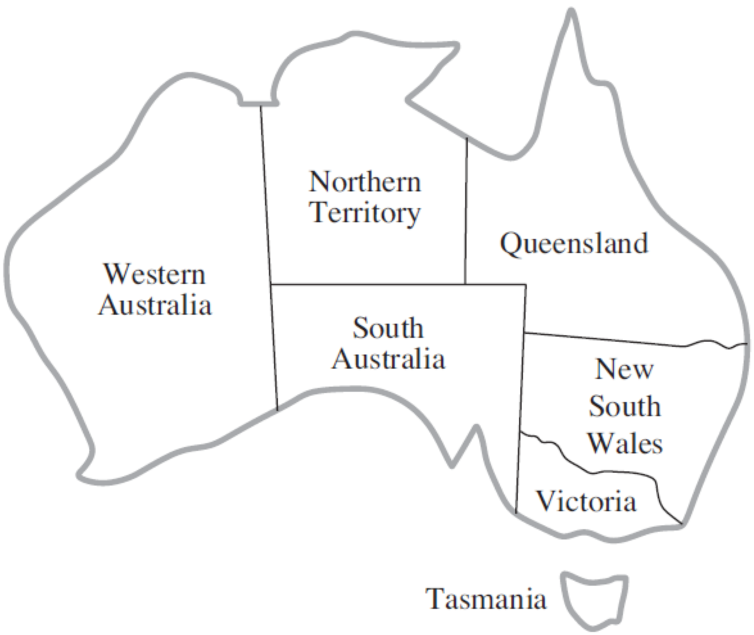
function AC-3(*csp*) **returns** false if an inconsistency is found and true otherwise
  inputs: *csp*, a binary CSP with components $(X,\ D,\ C)$
  local variables: *queue*, a queue of arcs, initially all the arcs in *csp*

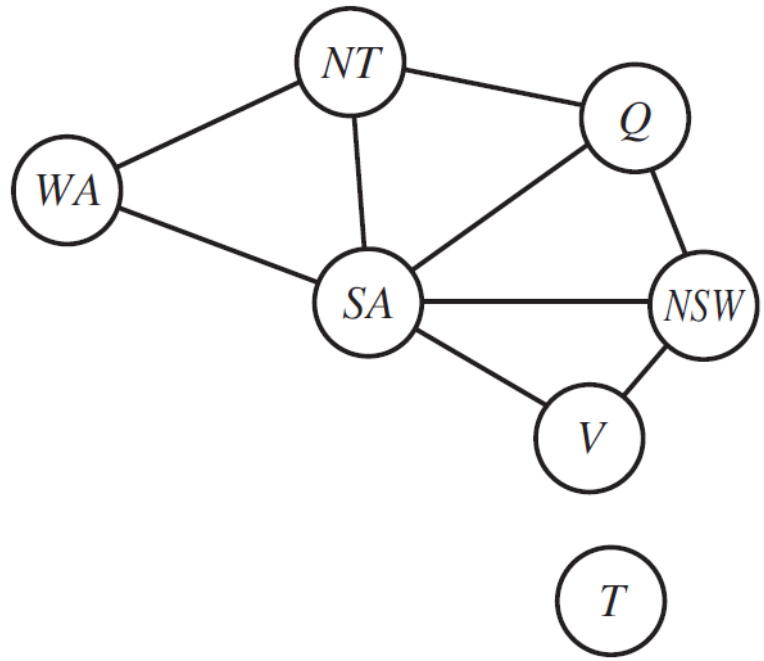
  while *queue* is not empty **do**
    $(X_i,\ X_j) \leftarrow$ REMOVE-FIRST(*queue*)
    if REVISE(*csp*, $X_i,\ X_j$) **then**
      if size of $D_i = 0$ **then return** *false*
      for each X_k **in** X_i.NEIGHBORS $\cancel{- \{X_j\}}$ **do**
        add $(X_k,\ X_i)$ to *queue*
  return *true*

---

function REVISE(*csp*, $X_i,\ X_j$) **returns** true iff we revise the domain of X_i
  revised $\leftarrow$ *false*
  for each x **in** D_i **do**
    if no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j **then**
      delete x from D_i
      revised $\leftarrow$ *true*
  return *revised*

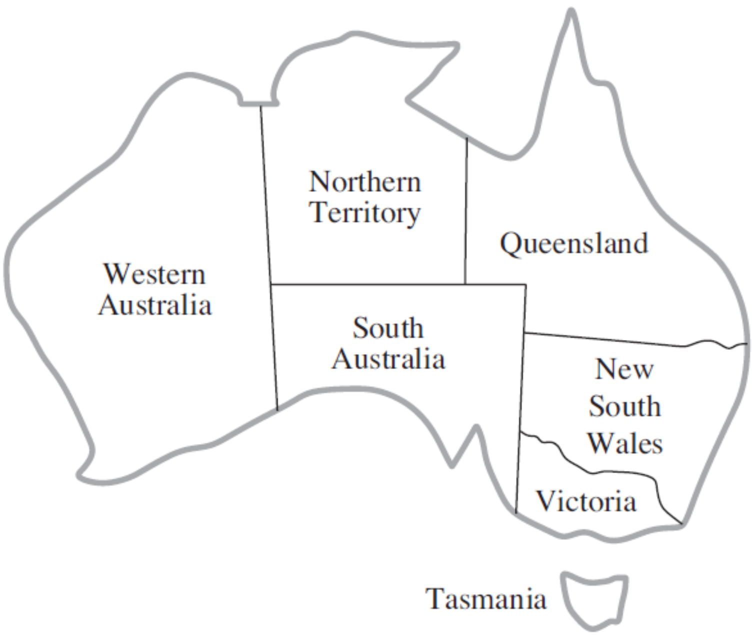
EXAMPLE: AC-3 ON THE AUSTRALIA MAP

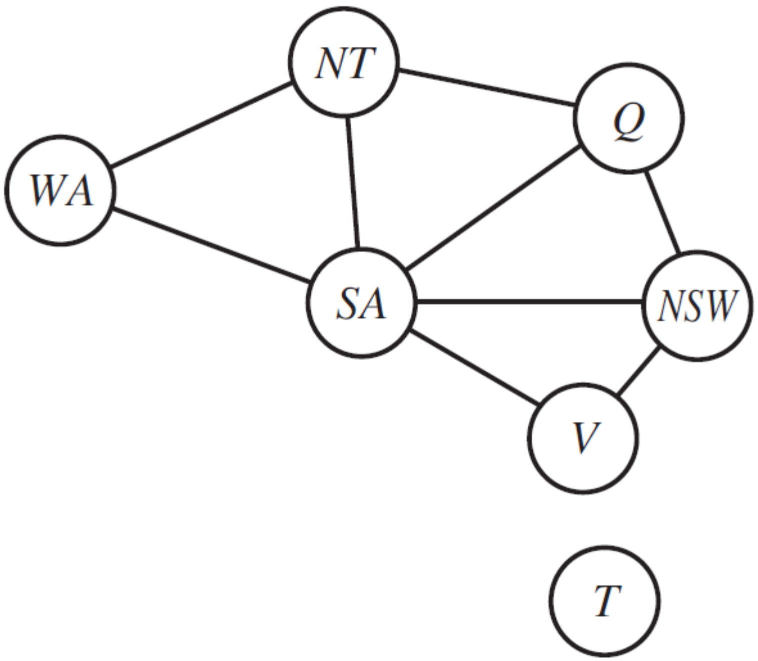




1. Assume $D_{WA} = \{R\}$, $D_V = \{G\}$
2. Assume $D_{WA} = \{R\}$, $D_{NSW} = \{G\}$

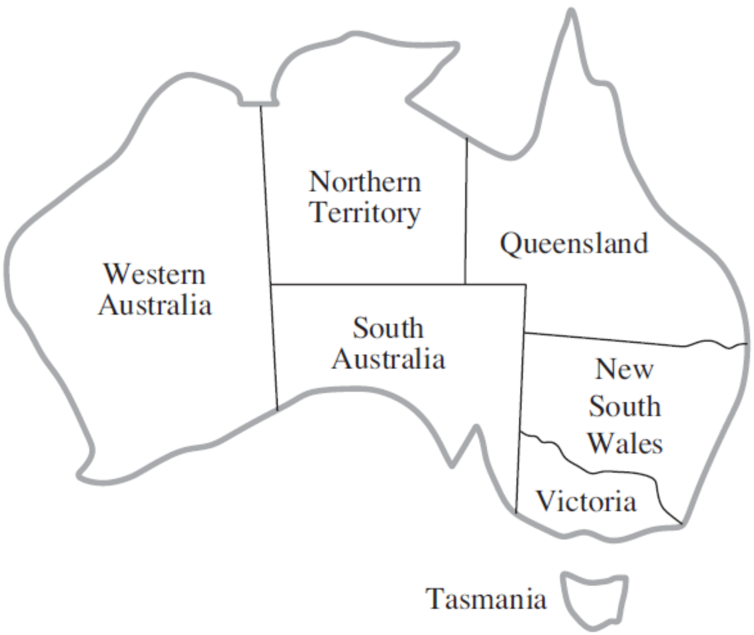
EXAMPLE: AC-3 ON THE AUSTRALIA MAP

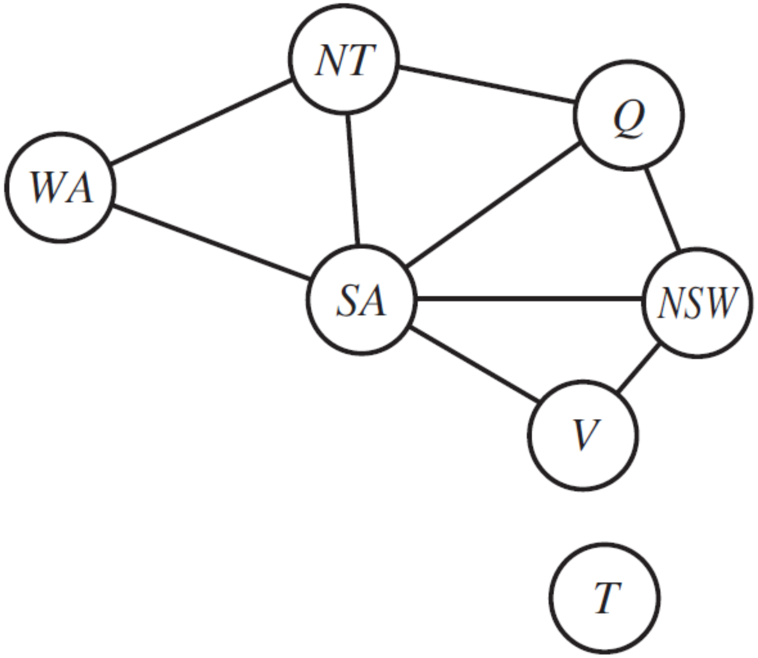




Assume $D_{WA} = \{R\}$, $D_V = \{G\}$

EXAMPLE: AC-3 ON THE AUSTRALIA MAP





Assume $D_{WA} = \{R\}$, $D_{NSW} = \{G\}$

More Constraint Propagation

- Arc consistency cannot detect all inconsistencies
- Consider coloring a map of three inter-connected states with two colors. They are arc-consistent.
- **Path consistency**
  - Triples of variables
  - A two-variable set $\{X_i, X_j\}$ is path-consistent with respect to a third variable X_m if, for every assignment $\{X_i=a, X_j=b\}$, there is an assignment to X_m that satisfies the constraints on $\{X_i, X_m\}$ and $\{X_m, X_j\}$.

More Constraint Propagation

- ***K*-consistency**
  - A CSP is k-consistent if, for any set of k-1 variables and for any consistent assignment to those variables, a consistent value can always be assigned to any k^{th} variable.
  - 1-consistency = node consistency
  - 2-consistency = arc consistency
  - 3-consistency = path consistency

Bounds Consistency

- A CSP is *bounds consistent* if for every variable X, and for every value between its lower and upper bounds, there exist some value for Y that satisfies the constraint between X and Y.
- Useful for variables with infinite or large domains
- Example
  - D_X = [0, 200], D_Y=[0, 300]; $X+Y = 400$. After making X and Y bounds consistent, what are D_X and D_Y?

Searching For Solutions

- Rather than treating all of the variables as a single state, we will treat the variables as the nodes of the search tree
- Depth-first search: intuitively (the formal version is in the next slide)
  - Pick an unassigned variable *var*
  - for each *value* in the domain of *var*
    - Assign *value* to the *var*
    - Propagate constraints
    - If inconsistency is detected, backtrack

Backtracking Search

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment then
          add {var = value} to assignment
          inferences ← INFERENCE(csp, var, value)
          if inferences ≠ failure then
              add inferences to assignment
              result ← BACKTRACK(assignment, csp)
              if result ≠ failure then
                  return result
      remove {var = value} and inferences from assignment
  return failure
```

Important Choices

1. Which variable to choose?
2. How to order its values?
3. What inference to use for propagating constraints?

Variable Ordering

- **Minimum remaining values (MRV) heuristic**
  - Also called "most-constrained" or "fail-first"
  - Prunes early
  - Might not help initially where domains are large
- **Degree heuristic**
  - Choose the variable that is involved in the greatest number of constraints with the remaining unassigned variables
  - A good tie-breaker for MRV

Value Ordering

- **Least-constraining value (LCV) heuristic**
  - Prefers the value that rules out the fewest choices for the neighboring variables
- MRV chooses the most-constrained variable and LCV chooses the least-constraining value. Why?

Interleaving Search and Inference

- **Forward checking**
  - When a variable X is assigned a value, make all the connected variables, Y, arc-consistent with X.
  - Fast but does not detect all inconsistencies
- **Maintaining arc consistency (MAC)**
  - When a variable X_i is assigned a value, call AC-3 with the queue = (X_j , X_i).

An Example

- The map coloring CSP with
  - MRV
  - Degree
  - Forward checking

Cryptarithmetic

- TWO + TWO = FOUR
- Every letter represents a different digit
- Let's solve an easier version, where F=1, O is less than 5, and W is less than 5, and 1 removed from every variables' domains
  - F=1 (already assigned)
  - O: {0, 2, 3, 4}
  - R: {0, 2, 3, 4, 5, 6, 7, 8, 9}
  - T: {0, 2, 3, 4, 5, 6, 7, 8, 9}
  - U: {0, 2, 3, 4, 5, 6, 7, 8, 9}
  - W: {0, 2, 3, 4}

Sudoku – AC-3 Can Solve This

6						2	1	
		8		1				4
	5	4	6					
		7			9	8	2	
	4			5	1	9		3
9			3	7			5	
		9		8				
3		5		2			6	
7	8							

Sudoku – AC-3 Can Almost Solve This

6						2	1	
		8		1				4
	5	4	6					
		7			9	8	2	
	4			5	1	9		3
9			3	7			5	
		9		8				
3		5		2			6	
7								

Sudoku – AC-3 Converts The Previous Problem Into This

6	9	3	8	4	7	2	1	5
2	7	8	5	1	3	6	9	4
1	5	4	6	9	2	3	8	7
5	3	7	4	6	9	8	2	1
8	4	6	2	5	1	9	7	3
9	{1,2}	{1,2}	3	7	8	4	5	6
4	{1,6}	9	{1,7}	8	{5,6}	{1,5,7}	3	2
3	{1,8}	5	{1,7,9}	2	4	{1,7}	6	{8,9}
7	{1,2,6,8}	{1,2}	{1,9}	3	{5,6}	{1,5}	4	{8,9}

Non-Binary to Binary

- $a+b=2*c$
- $c+d=12$
- $a = \{0,1,2,3,4\}$
- $b = \{0,1,2\}$
- $c = \{0,1,2,3,4\}$
- $d = \{0..9\}$
- All are distinct

LOCAL SEARCH

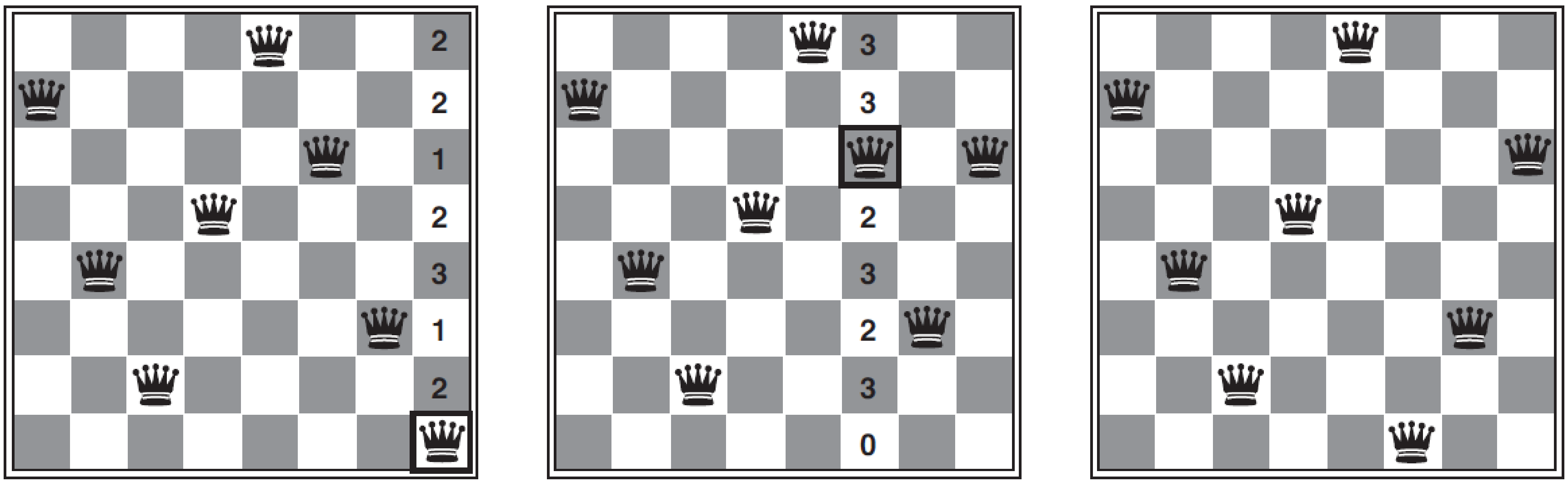
- Start with an initial guess
- Pick a variable that violates constraints and choose a value for it
- Min-conflicts heuristic
  - Pick a value that results in the minimum number of conflicts with the other variables

MIN-CONFLICTS

function MIN-CONFLICTS(*csp*, *max_steps*) **returns** a solution or failure
  inputs: *csp*, a constraint satisfaction problem
    max_steps, the number of steps allowed before giving up

  current ← an initial complete assignment for *csp*
  for *i* = 1 to *max_steps* **do**
    if *current* is a solution for *csp* **then return** *current*
    var ← a randomly chosen conflicted variable from *csp*.VARIABLES
    value ← the value *v* for *var* that minimizes CONFLICTS(*var*, *v*, *current*, *csp*)
    set *var* = *value* in *current*
  return *failure*

Min-Conflicts Example



So Far

- Agent-based modeling – Chapter 2
- Search for problem solving
  - Goal-based – Chapter 3
    - DFS, BFS, …, A*
  - Utility-based – Chapter 5
    - Minimax, alpha-beta
  - Constraint satisfaction – Chapter 6
    - Backtracking

Next

- Knowledge Representation and Reasoning – Logic
  - Chapters 7, 8, 9