

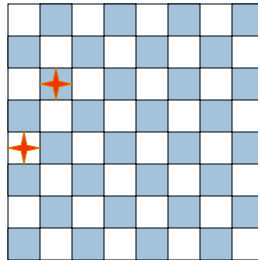
CONSTRAINT SATISFACTION

Today

- Reading
 - AIMA Chapter 6

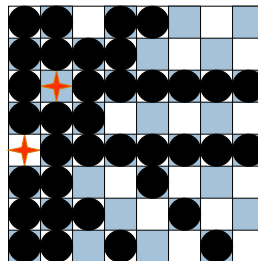
- Goals
 - Constraint satisfaction problems (CSPs)
 - Types of CSPs
 - Inference
 - Search + Inference

8-queens problem



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

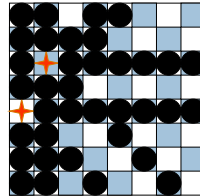
8-queens problem



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

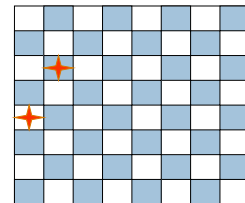


Constraint satisfaction problems

- **Set of variables** $\{X_1, X_2, \dots, X_n\}$
- Each variable X_i has a **domain** D_i of possible values
- **Set of constraints** $\{C_1, C_2, \dots, C_p\}$
 - ▣ 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?

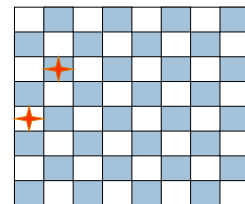


Example: 8-queens problems

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

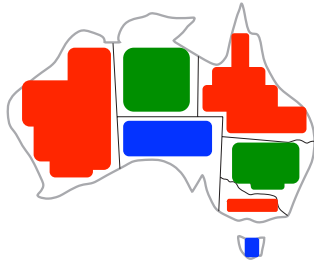
$$X_i = k \implies X_j \neq k \quad \forall i \neq j$$

$$X_i = k_i, X_j = k_j \implies |i - j| \neq |k_i - k_j|$$



Example: Map coloring

- **Variables:** {WA, NT, SA, Q, NSW, V, T}
- **Domains:** {red, blue, green}
- **Constraints:** adjacent regions have different colors
 - ▣ Implicit: $WA \neq NT, WA \neq SA, SA \neq NT, NT \neq Q, \dots$
 - ▣ Explicit: $(WA, NT) \in \{(red, green), (red, blue), \dots\}$



Example: Task scheduling

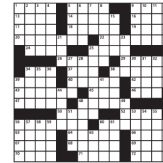
- **Variables:** {AxleF, AxleB, WheelRF, WheelLF, ..., Inspect}
- **Domains:** Time task starts $D = [0, 1, 2, \dots, \infty)$
- **Constraints:**
 - ▣ Axle must be done before the wheel
 - $AxleF + 10 < WheelLF$
 - $AxleF + 10 < WheelRF$
 - ▣ The front axle and the back axle cannot be done at the same 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

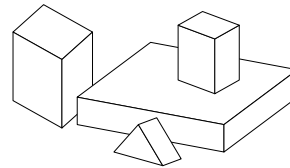
- sudoku, cryptarithmic

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



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

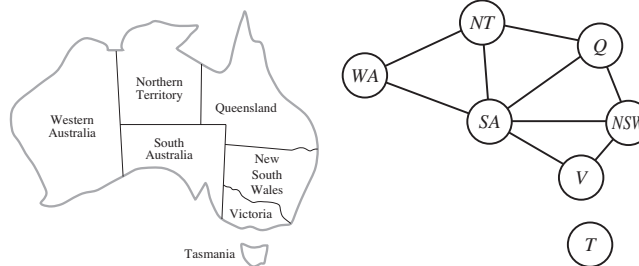
- linear programming problems with linear equality or inequality constraints solvable in polynomial time

Types of CSPs - constraints

- **Unary constraints** involve a single variable
 - e.g. SA \neq green
- **Binary constraints** involve pairs of variables
 - SA \neq 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



Solving CSPs: 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
  ( $X_i, X_j$ ) ← REMOVE-FIRST(queue)
  if REMOVE-INCONSISTENT-VALUES( $X_i, X_j$ )
    if size  $D_i$  == 0 return false
    for each arc ( $X_k, X_i$ )
      add ( $X_k, X_i$ ) to queue
return true
  
```

function REMOVE-INCONSISTENT-VALUES(X_i, X_j)

```

revised ← false
for each x in  $D_i$ 
  if  $\nexists$  y in  $D_j$  s.t. (x,y) satisfies constraints
    delete x from  $D_i$ 
  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
    ( $X_i, X_j$ ) ← REMOVE-FIRST(queue)
    if REMOVE-INCONSISTENT-VALUES( $X_i, X_j$ )
      if size  $D_i$  == 0 return false
      for each arc ( $X_k, X_i$ )
        add ( $X_k, X_i$ ) to queue
  return true
  
```

c constraints (arcs)
 d domain size

$O(cd)$

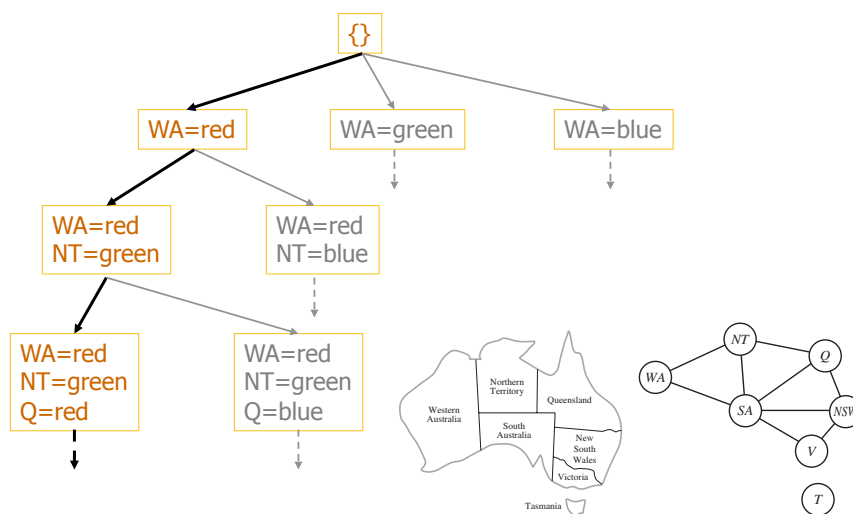
Total
 $O(cd^3)$

```

function REMOVE-INCONSISTENT-VALUES( $X_i, X_j$ )
  revised ← false
  for each  $x$  in  $D_i$ 
    if  $\nexists y$  in  $D_j$  s.t. ( $x, y$ ) satisfies constraints
      delete  $x$  from  $D_i$ 
    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

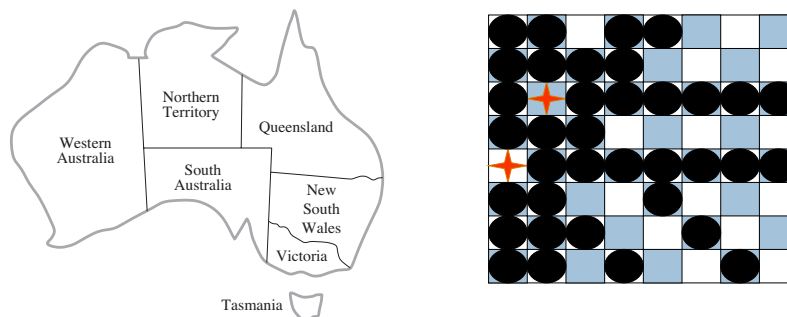
- Backtracking = DFS + variable ordering + constraints checking
- **Idea 1: One variable at a time**
 - Variable assignments are commutative so fix ordering
 - i.e. (WA = red, NT = green) is the same as (NT = green, WA = red)
- **Idea 2: Check constraints as we go**
 - Consider only values which do not conflict with previous assignments
 - May take some computation to check
 - “incremental goal test”
 - (Additional inference is optional, e.g. forward checking or arc-consistency)

Improving Backtracking search

- **Idea 1: Intelligent ordering**
 - ▣ Which variable X should be assigned a value next?
 - ▣ In which order should its domain D be sorted?
- **Idea 2: Incorporating inference**
 - ▣ Forward checking
 - ▣ AC-3
- **Idea 3: Exploiting structure**
 - ▣ Can we exploit the problem structure?

Idea 1: Intelligent Ordering

- Which variable should we choose?



Idea 1: Intelligent Ordering

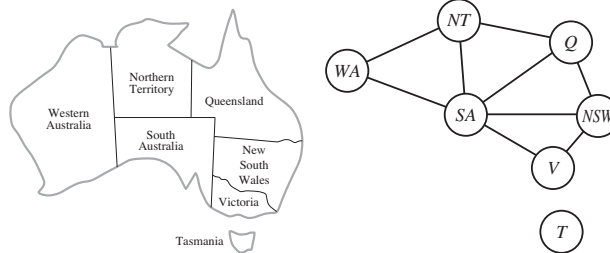
- **Variable ordering**
 - **Minimum-remaining values heuristic** - Choose the variable with the fewest “legal” moves remaining
 - **Degree heuristic** - Choose variable involved in the largest number of constraints with remaining unassigned variables
- **Value ordering**
 - **Least-constraining value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

Idea 2: Incorporating Inference

- **Forward checking**
 - After an assignment $X = x$, ensure all arcs of the form (Y,X) are arc consistent
- **Run AC-3 algorithm**
 - Ensure all arcs are arc consistent
- **Run path-consistency or k-consistency algorithm**

Example

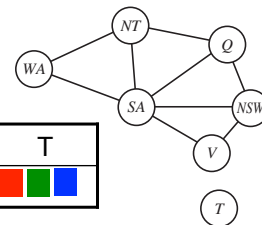
- Run Backtracking on graph coloring
 - ▣ Use MRV heuristics to choose variable
 - ▣ Use LCV heuristics to choose value
 - ▣ Use forward checking for inference



Limitations of Forward Checking

- Backtracking + Forward checking
 - ▣ WA, Q, V, NT, NSW, SA, T

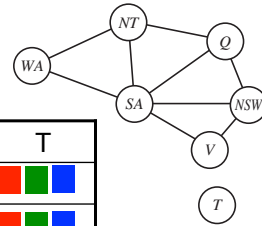
WA	NT	Q	NSW	V	SA	T
■ ■ ■	■ ■ ■	■ ■ ■	■ ■ ■	■ ■ ■	■ ■ ■	■ ■ ■



Limitations of Forward Checking

□ Backtracking + Forward checking

- WA, Q, V, NT, NSW, SA, T

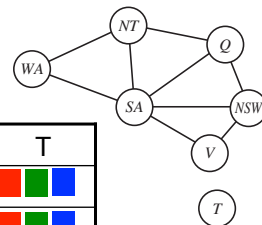


WA	NT	Q	NSW	V	SA	T
■ ■ ■	■ ■ ■	■ ■ ■	■ ■ ■	■ ■ ■	■ ■ ■	■ ■ ■
■	■ ■	■ ■ ■	■ ■ ■	■ ■ ■	■ ■	■ ■ ■
■	■ ■	■ ■ ■	■ ■ ■	■ ■ ■	■ ■	■ ■ ■

Limitations of Forward Checking

□ Backtracking + Forward checking

- WA, Q, V, NT, NSW, SA, T

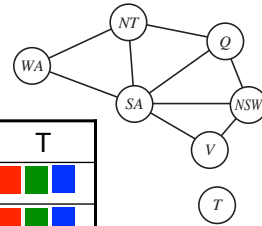


WA	NT	Q	NSW	V	SA	T
■ ■ ■	■ ■ ■	■ ■ ■	■ ■ ■	■ ■ ■	■ ■ ■	■ ■ ■
■	■ ■	■ ■ ■	■ ■ ■	■ ■ ■	■ ■	■ ■ ■
■	■ ■	■ ■ ■	■ ■ ■	■ ■ ■	■ ■	■ ■ ■
■	■ ■	■ ■ ■	■ ■ ■	■ ■ ■	■ ■	■ ■ ■

Limitations of Forward Checking

□ Backtracking + Forward checking

- WA, Q, V, NT, NSW, SA, T

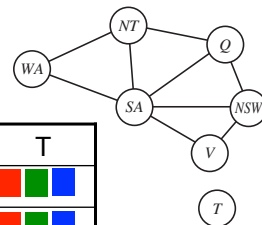


WA	NT	Q	NSW	V	SA	T
Red Green Blue	Red Green Blue	Red Green Blue	Red Green Blue	Red Green Blue	Red Green Blue	Red Green Blue
Red	Green Blue	Red Green Blue	Red Green Blue	Red Green Blue	Green Blue	Red Green Blue
Red	Blue	Green	Red Blue	Red Green Blue	Blue	Red Green Blue
Red	Blue	Green	Red	Blue		Red Green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

Limitations of Forward Checking

□ Backtracking + Forward checking

- WA, Q, V, NT, NSW, SA, T



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
Red	Blue	Green	Red Blue	Red Green Blue	Blue	Red Green Blue
Red	Blue	Green	Red	Blue		Red Green Blue

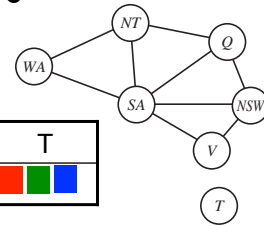
Could have detected earlier that things were going wrong!

Limitations of Forward Checking

- Using AC-3 in stead of forward checking

- ▣ WA, Q, V, NT, NSW, SA, T

WA	NT	Q	NSW	V	SA	T
■ ■ ■	■ ■ ■	■ ■ ■	■ ■ ■	■ ■ ■	■ ■ ■	■ ■ ■

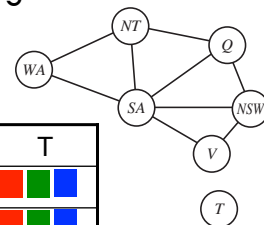


Limitations of Forward Checking

- Using AC-3 in stead of forward checking

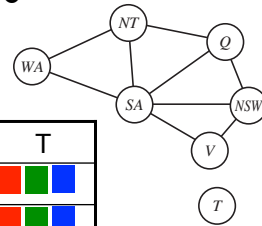
- ▣ WA, Q, V, NT, NSW, SA, T

WA	NT	Q	NSW	V	SA	T
■ ■ ■	■ ■ ■	■ ■ ■	■ ■ ■	■ ■ ■	■ ■ ■	■ ■ ■
■	■ ■	■ ■ ■	■ ■ ■	■ ■ ■	■ ■	■ ■ ■



Limitations of Forward Checking

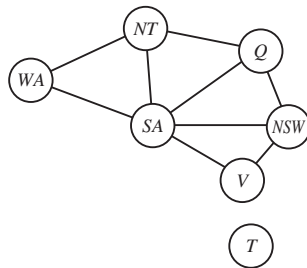
- Using AC-3 in stead of forward checking
 - ▣ WA, Q, V, NT, NSW, SA, T



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
Red	Blue	Green			X	

Idea 3: Exploit Structure

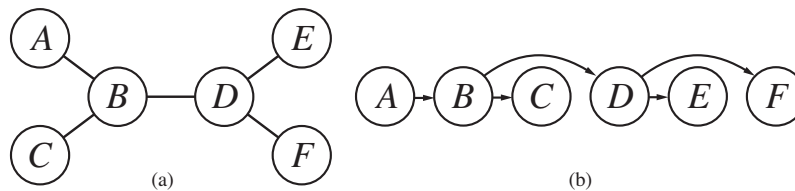
- Independent subproblems
 - ▣ Find connected components of the constraint graph
 - ▣ e.g. Tasmania and the mainland are independent
 - ▣ If we can split n variables into c subproblems of n/c variables each: $O(d^n) \rightarrow O(d^{c n/c})$



Tasmania is independent of the mainland!

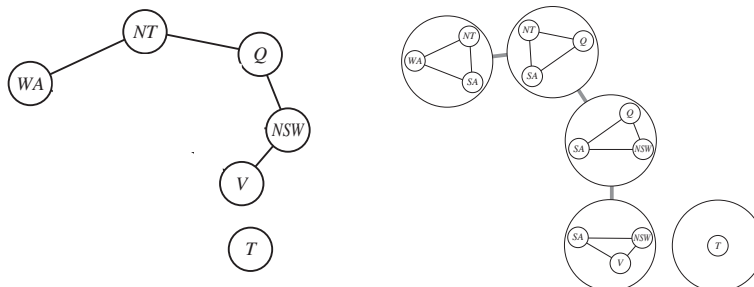
Idea 3: Exploit Structure

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



Idea 3: Exploit Structure

- 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 + fixed ordering + constraints checking
 - ▣ General (not problem-specific) heuristics
- Improving Backtracking
 - ▣ Intelligent ordering
 - ▣ Incorporating inference
 - ▣ Exploiting structure