Constraint Satisfaction

CS 486/686: Introduction to Artificial Intelligence
Fall 2013

Outline

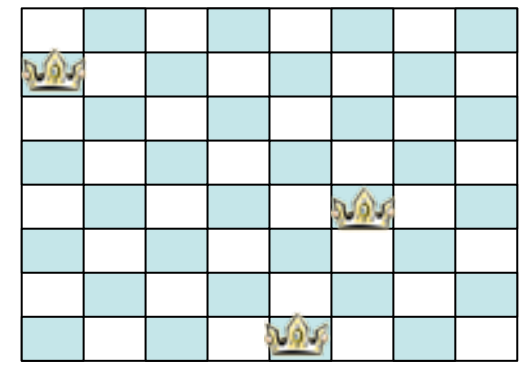
- What are Constraint Satisfaction Problems (CSPs)?
- Standard Search and CSPs
- Improvements
  - Backtracking
  - Backtracking + heuristics
  - Forward Checking

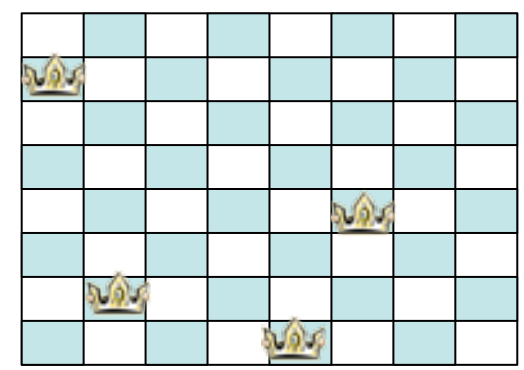
Introduction

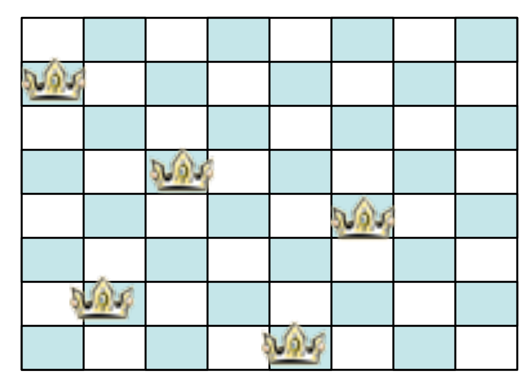
- We have been solving problems by searching in a **space of states**
  - Treating states as black boxes
- Today, we study problems where **state structure** is important

Queens Problem

- **States**: All arrangements of 0,1,..., or 8 queens
- **Initial state**: 0 queens on the board
- **Successor function**: Add a queen to the board
- **Goal Test**: 8 queens on the board with no two attacking each other





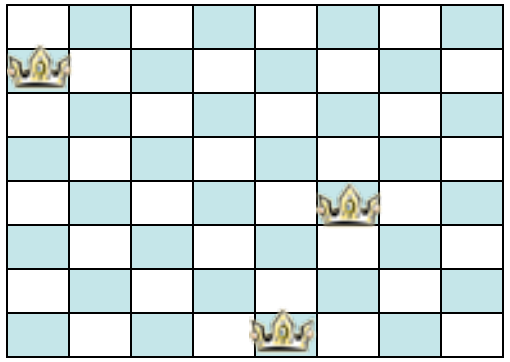


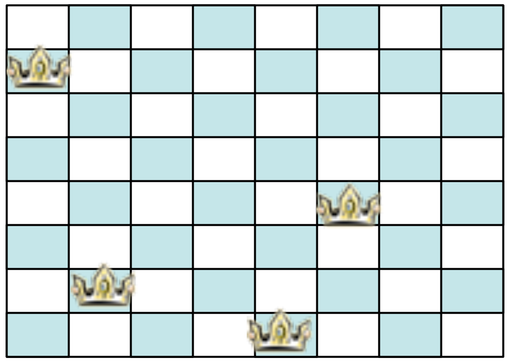
$64 \times 63 \times \ldots 53 \sim 3 \times 10^{14}$ states

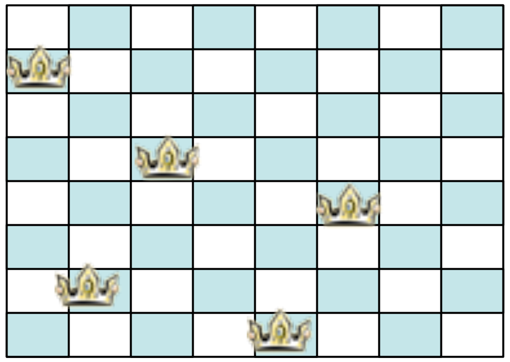
Queens Problem

- **States**: All arrangements of k queens (0≤k≤8), one per column in the leftmost k columns, with no queen attacking another
- **Initial state**: 0 queens on the board
- **Successor function**: Add queen leftmost empty column such that it is not attacked
- **Goal Test**: 8 queens on the board

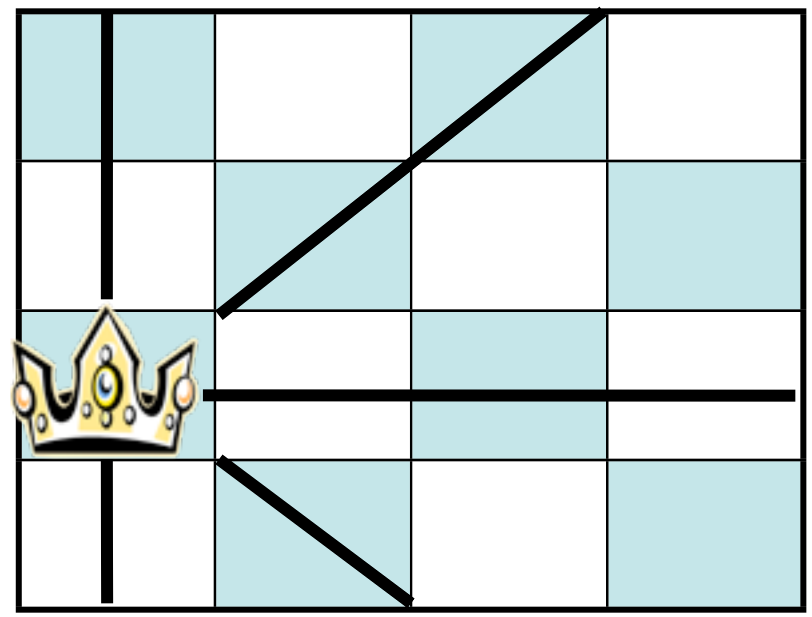
2057 states







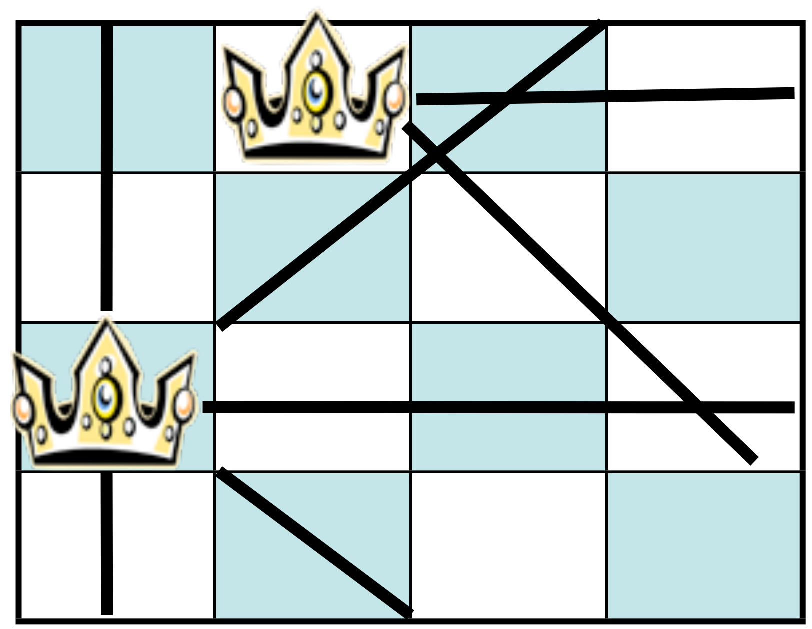
4 Queens: Constraint Propagation



Place a queen in a square

Remove conflicting squares from consideration

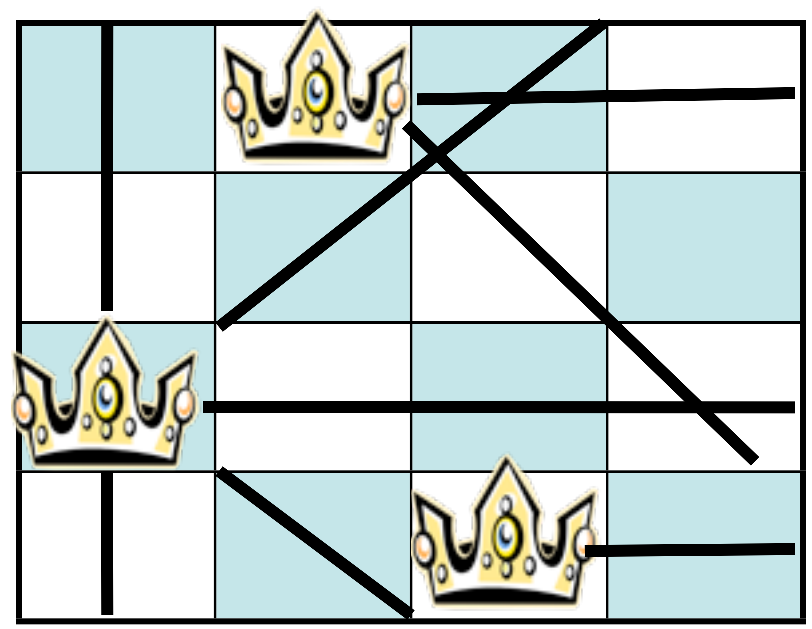
4 Queens: Constraint Propagation



Place a queen in a square

Remove conflicting squares from consideration

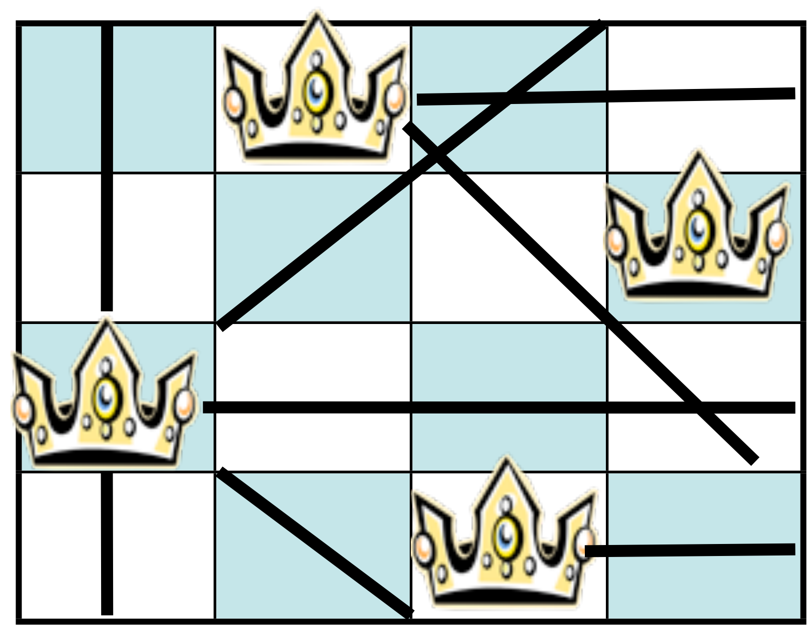
4 Queens: Constraint Propagation



Place a queen in a square

Remove conflicting squares from consideration

4 Queens: Constraint Propagation



Place a queen in a square

Remove conflicting squares from consideration

CSP Definition

- A constraint satisfaction problem (CSP) is defined by {V, D,C} where
  - $V=\{V_1,V_2,...,V_n\}$ is the set of **variables**
  - $D=\{D_1,D_2,...,D_n\}$ is the set of **domains**
  - $C=\{C_1,C_2,...,C_m\}$ is the set of **constraints**

CSP Definition

- A state is an **assignment** of values to some (or all) variables
  - $\{V_i=x_i, V_j=x_j,\ldots\}$
- **Consistent** assignments
  - No violated constraints
- A **solution** is a complete, consistent assignment

Example: 8 Queens

- 64 variables V_{ij}, i=1..8, j=1..8
- Domain $D_{ij}=\{0,1\}$
- Constraints
  - $V_{ij}=1 \Rightarrow V_{ik}=0$ for $j \neq k$
  - $V_{ij}=1 \Rightarrow V_{kj}=0$ for $i \neq k$
  - Similar constraints for diagonals
  - $\sum_{ij} V_{ij}=8$

Binary Constraints:
Relate two variables

Example: 8 Queens

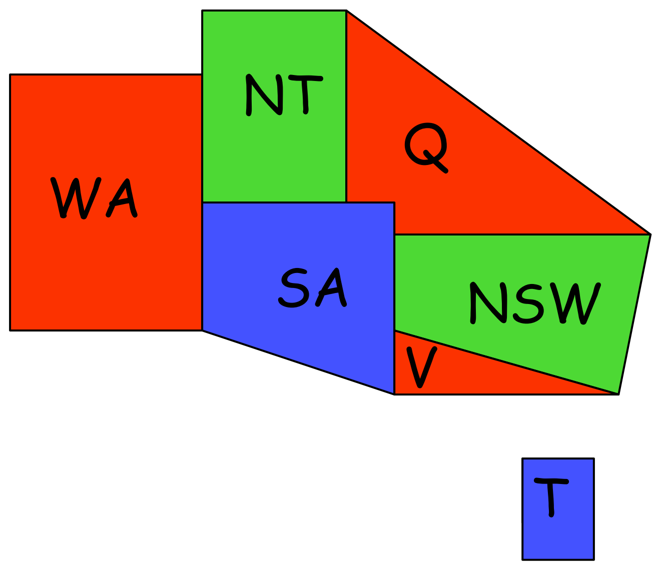
- 8 variables, V_i $i=1..8$
- Domains $D_i=\{1,2,...,8\}$
- Constraints
  - $V_i=k \Rightarrow V_j \neq k$ for all $j \neq i$
  - Similar constraints for diagonals

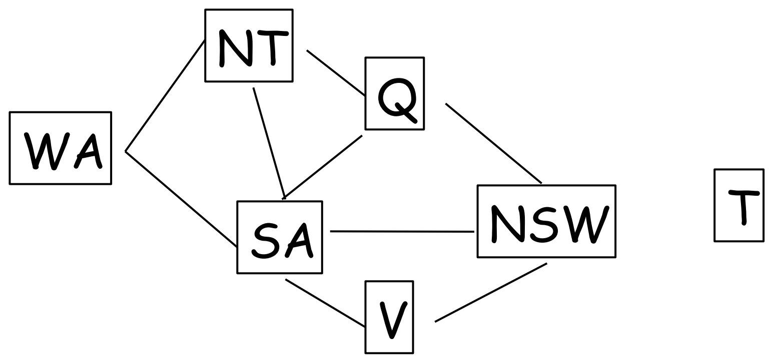
Example: Map Colouring

- 7 variables {WA, NT, SA, Q, NSW, V, T}

- Each variable has same domain {red, blue, green}

- No two adjacent variables have the same value





Constraint graph

Example: 3 Sat

- n Boolean variables $V_1,...,V_n$
- K constraints of the form $V_i^* \vee V_j^* \vee V_k^*$ where V_i^* is either V_i or $\sim V_i$
- NP-Complete

Variable Types of CSPs

- Discrete and finite
- Discrete variables with infinite domains
  - Constraint languages
- Continuous domains
  - Linear programming etc

Types of Constraints

- Unary
  - restricts a variable to a single value
- Binary
  - Constraint graph
- Higher order constraints involve three or more variables
  - Constraint hypergraphs

CSPs and Search

- n variables $V_1,..,V_n$
- **Valid assignment**: $\{V_1=x_1,...,V_k=x_k\}$ for $0 \leq k \leq n$ such that values satisfy constraints
- **States**: valid assignments
- **Initial state**: empty assignment
- **Successor**: $\{V_1=x_1,...V_k=x_k\} \rightarrow \{V_1=x_1,...,V_k=x_k,V_{k+1}=x_{k+1}\}$
- **Goal test**: complete assignment
  - If all domains have size d then there are $O(d^n)$ complete assignments

Communtativity

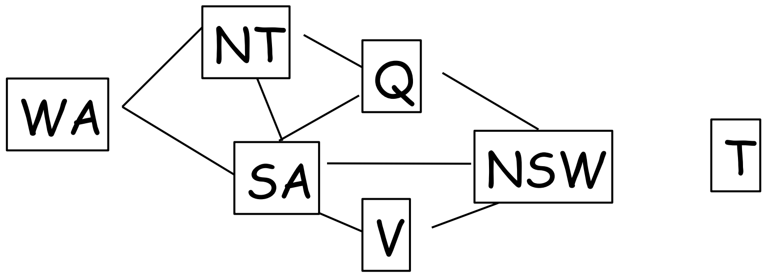
- CSPs are commutative
- Order of actions taken does not effect outcome
  - Can assign values to variables in any order
- CSP algorithms take advantage of this
  - Consider possible assignments for a **single variable at each node in search tree**

Backtracking Search

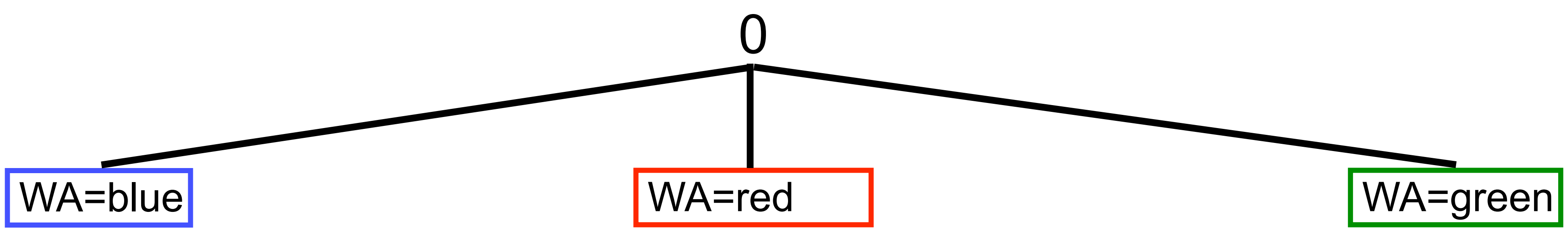
- Select unassigned variable X
- For each value $\{x_1, \ldots, x_n\}$ in domain of X
  - If value satisfies constraints, assign $X=x_i$ and exit loop
- If an assignment is found
  - Move to next variable
- If no assignment found
  - Back up to preceding variable and try a different assignment for it

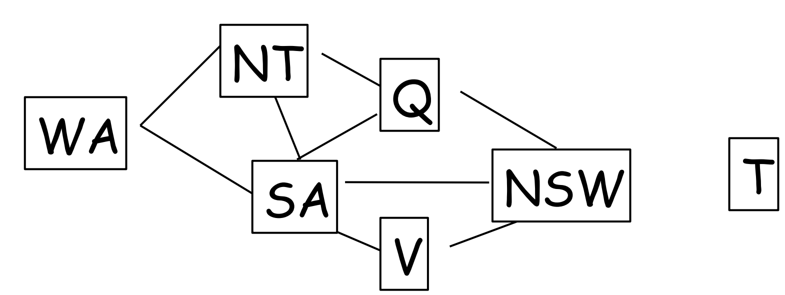
Backtracking Example

0

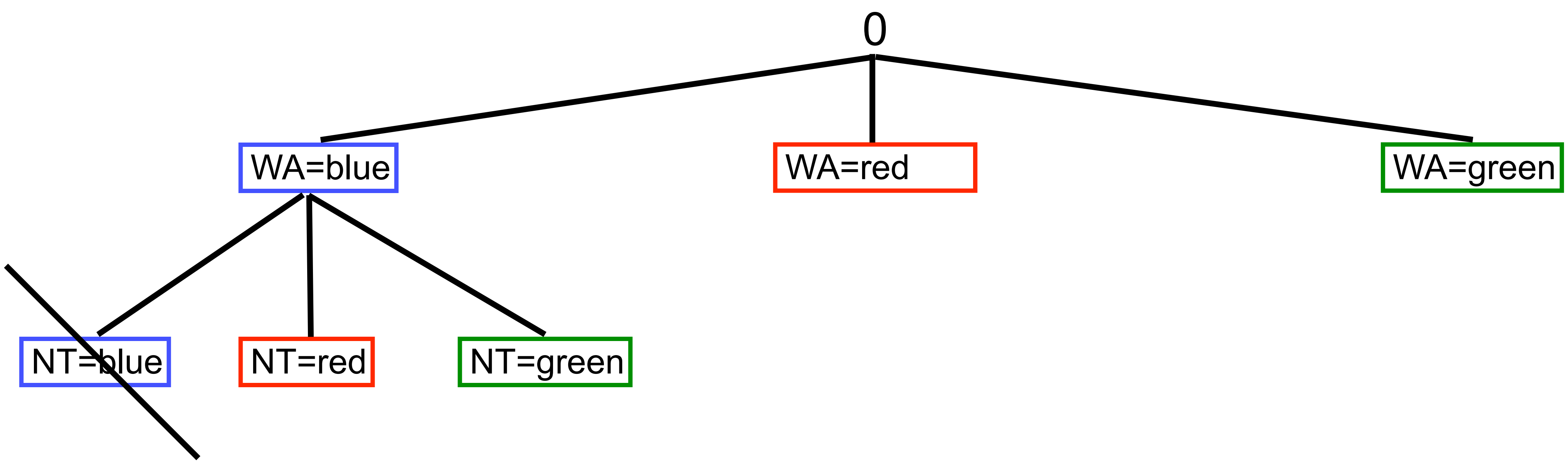


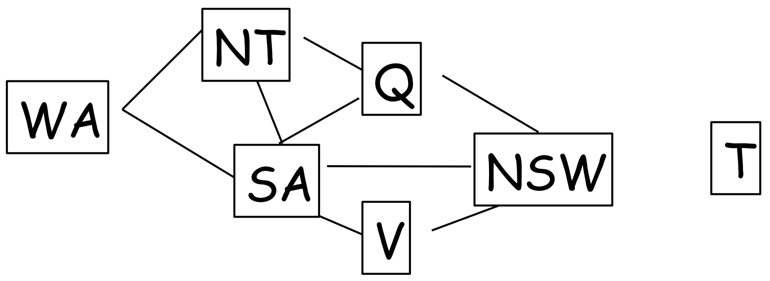
Backtracking Example



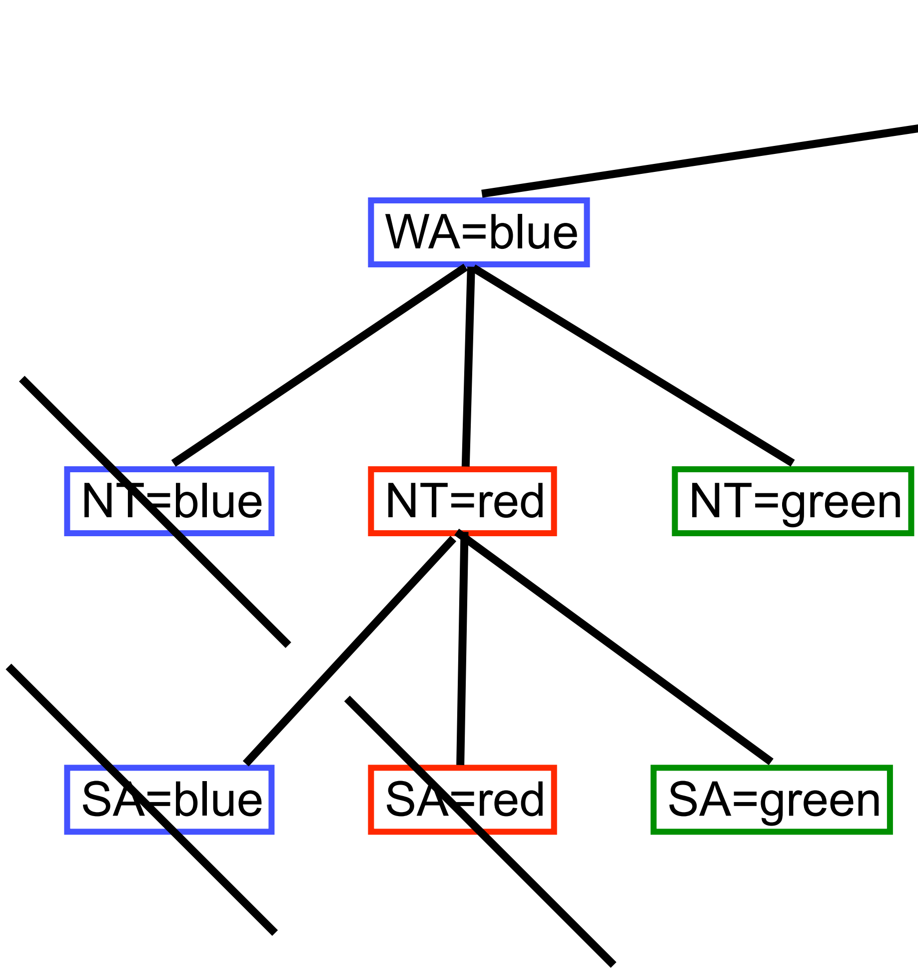


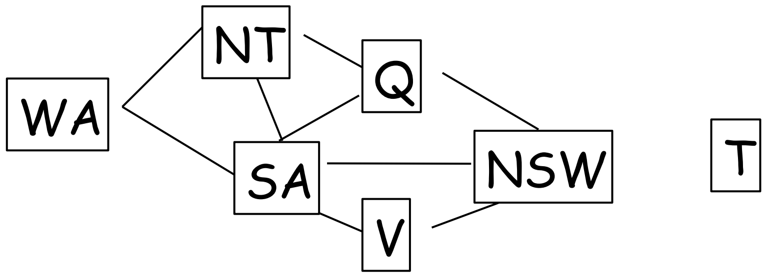
Backtracking Example





Backtracking Example





Backtracking and Efficiency

- Backtracking search = uninformed search method
  - Not very efficient
- Can we do better? Heuristics!
  - Which variable should be assigned next?
  - In which order should its values be tried?
  - Can we detect inevitable failure early?

Most Constrained Variable

- Choose the variable which has the fewest “legal” moves
  - AKA **minimum remaining values** (MRV)

D_{NT}={green, blue}

D_{SA}={green, blue}

D_{others}={red, green, blue}

D_{SA}={blue}

D_{Q}={blue, red}

D_{others}={red,green,blue}

Most Constraining Variable

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

SA is involved in 5 constraints

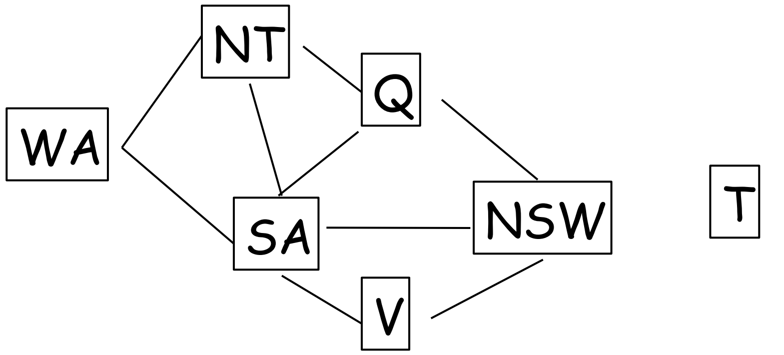
Least-Constraining Value

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

Forward Checking

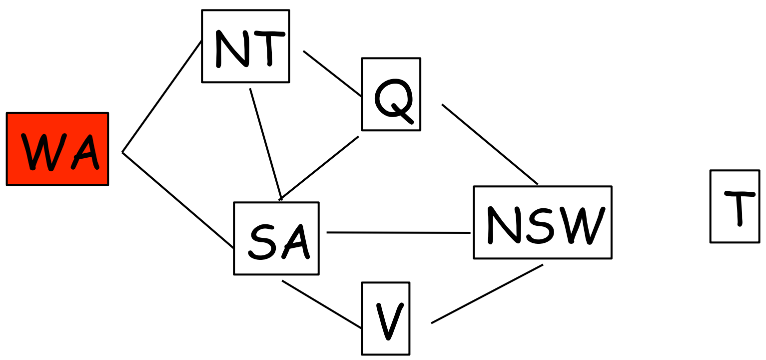
- Is there a way to detect failure early?
- Forward checking:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values

Example: Forward Checking



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB

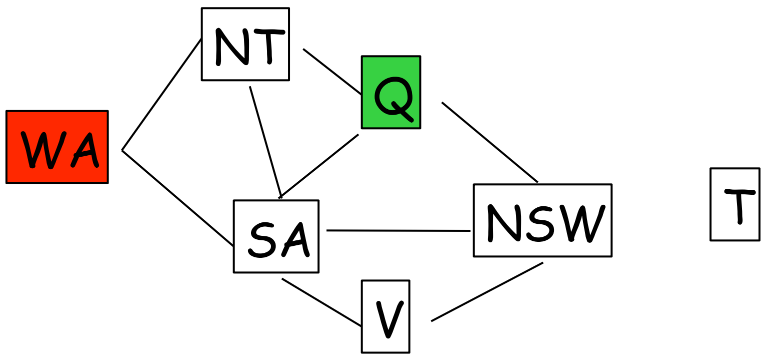
Example: Forward Checking



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB

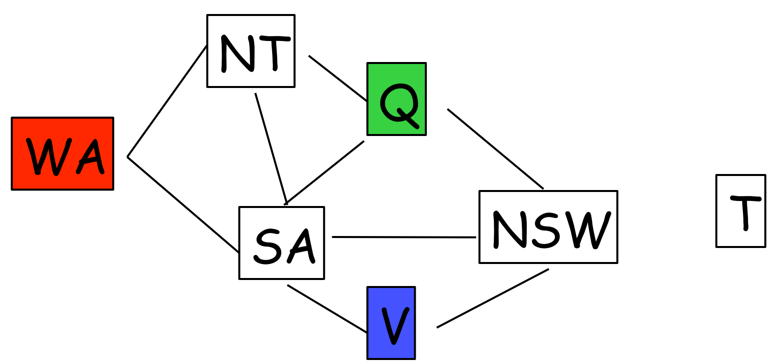
Forward checking removes the value Red of NT and of SA

Example: Forward Checking



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	GB	G	RGB	RGB	GB	RGB

Example: Forward Checking



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	B	G	RB	RGB	B	RGB
R	B	G	~~R~~B	B	~~B~~	RGB

Example: Forward Checking

Empty set: the current assignment
$\{(WA \leftarrow R), (Q \leftarrow G), (V \leftarrow B)\}$
does not lead to a solution

WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	B	G	RB	RGB	B	RGB
R	B	G	~~R~~B	B	~~B~~	RGB

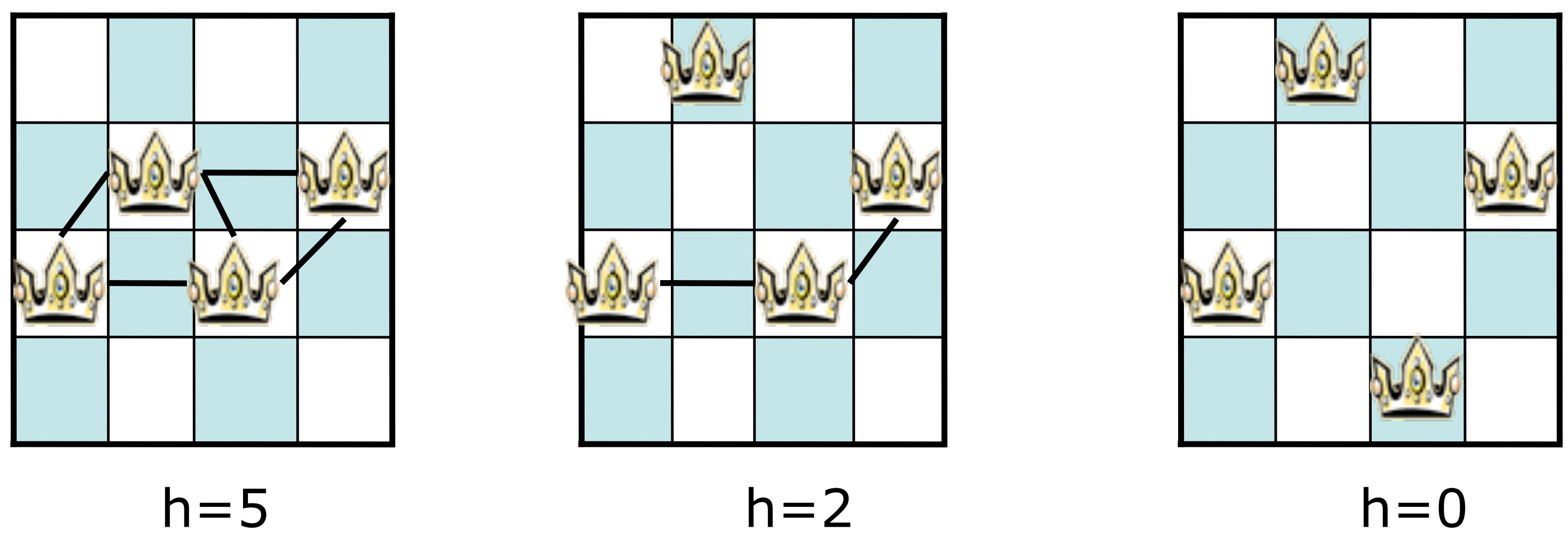
Arc Consistency

- Arc Consistency
  - Fast method of constraint propagation
  - Stronger than forward checking

Iterative Improvement Algorithms

- Start with a broken but complete assignment of values to variables
  - Allow states to have variable assignments that do not satisfy constraints
- Randomly select conflicted variables
- Operators reassign values
  - **Min-conflict heuristic:** choose value that violates fewest constraints

Example: 4 Queens



Given random initial state, can solve n-queens problem in almost constant time for arbitrary n with high probability (e.g. $n=10^7$)

Appears to be true for any randomly-generated CSP except in narrow range of ratio:

R=(number constraints)/(number variables)

Summary

- How to formalize problems as CSPs
- Backtracking search
- Heuristics
  - variable ordering
  - value ordering
  - forward checking
  - arc consistency
- Iterative Improvement approaches