NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Introduction to Data Science and Artificial Intelligence

Constraint Satisfaction and Game Playing

Assoc Prof Bo AN

Research area: artificial intelligence, computational game theory, optimization

www.ntu.edu.sg/home/boan

Email: boan@ntu.edu.sg

Office: N4-02b-55

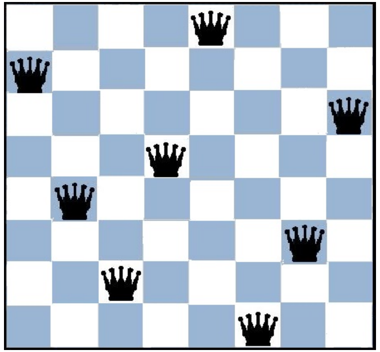
Lesson Outline

- Constraint Satisfaction
  - Backtracking search
  - Forward checking and constraint propagation
- Game Playing
  - Games as search problems
  - Minimax search strategy
  - Evaluation functions

Constraint Satisfaction Problem (CSP)

Goal: discover some state that satisfies a given set of constraints

Example: 8-Queens Problem



Example: Cryptarithmetic Puzzle

$$\begin{array}{r} S\ E\ N\ D \\ +\ M\ O\ R\ E \\ \hline M\ O\ N\ E\ Y \end{array}$$

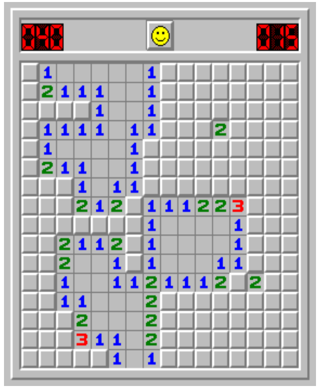
Constraint Satisfaction Problem (CSP)

Goal: discover some state that satisfies a given set of constraints

Example: Sudoku

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

Example: Minesweeper



Examples: Real-world CSPs

- Assignment problems
  - e.g. who teaches what class
- Timetabling problems
  - e.g. which class is offered when and where?
- Hardware configuration
- Transportation scheduling
- Factory scheduling
- Floor-planning

CSP

State

- defined by variables V_i with values from domain D_i

Example: 8-queens

- Variables: locations of each of the eight queens
- Values: squares on the board

Goal test

- a set of constraints specifying allowable combinations of values for subsets of variables

Example: 8-queens

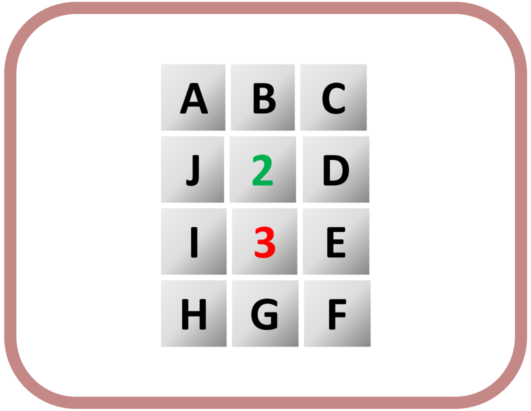
- Goal test: No two queens in the same row, column or diagonal

Example: Cryptarithmetic Puzzle

```
   S E N D
 + M O R E
 ---------
 M O N E Y
```

- Variables: D, E, M, N, O, R, S, Y
- Domains: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
- Constraints
  - $Y = D + E$ *or* $Y = D + E - 10$, etc.
  - $D \neq E$, $D \neq M$, $D \neq N$, etc.
  - $M \neq 0$, $S \neq 0$ (unary constraints: concern the value of a single variable)

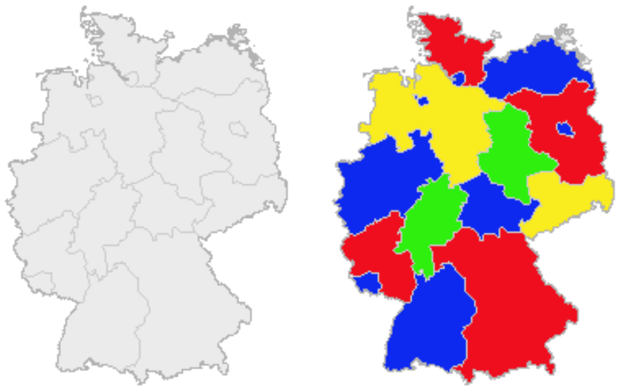
Example: Minesweeper



- Variables: The cells
- Domains: {0; 1} representing {safe, mined}
- Constraints: Each cell has a number $m \in \{ 1, \ldots , 8\}$ indicating the number of mines nearby, so m is equal to sum of value of neighbour cells

Example: Map Colouring

Colour a map so that no adjacent parts have the same colour



- Variables: Countries *Ci*
- Domains: {*Red, Blue, Green*}
- Contraints: C1 ≠ C2, C1 ≠ C5, etc.
  - binary constraints

Some Definitions

- A state of the problem is defined by an **assignment** of values to some or all of the variables.
- An assignment that does not violate any constraints is called a **consistent** or **legal** assignment.
- A **solution** to a CSP is an assignment with every variable given a value (**complete**) and the assignment satisfies all the constraints.

Applying Standard Search

- States: defined by the values assigned so far
- Initial state: all variables unassigned
- Actions: assign a value to an unassigned variable
- Goal test: all variables assigned, no constraints violated

Applying Standard Search

Question: How to represent constraints?

Answer: Explicitly (e.g., D ≠ E)

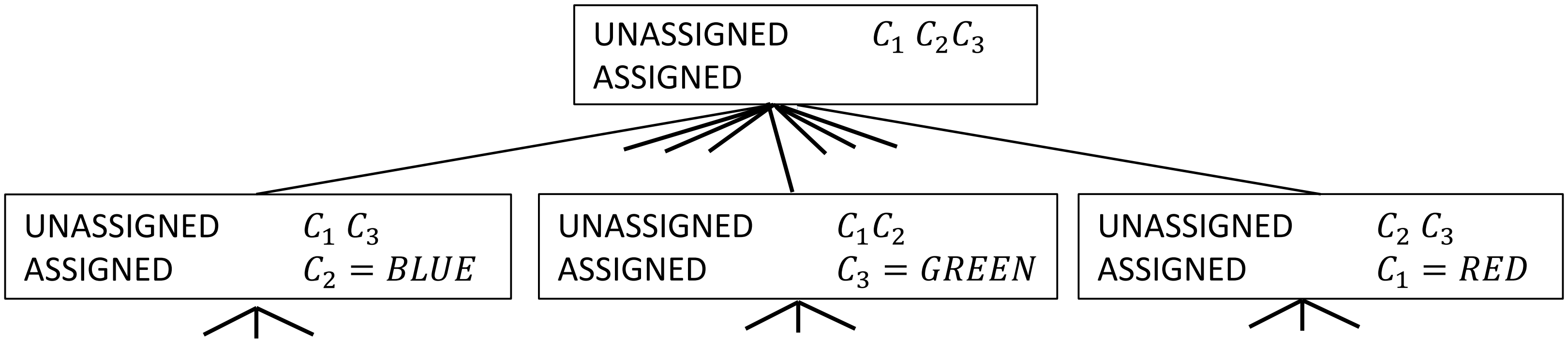
Example

- Row the 1st queen occupies: $V_1 \in \{1, 2, 3, 4, 5, 6, 7, 8\}$
  (similarly, for V_2)
- No-attack constraint for V_1 and V_2:
  { <1, 3> , <1, 4>, <1, 5>, … , <2, 4>, <2, 5>, …}

Implicitly: use a function to test for constraint satisfaction

Applying Standard Search...

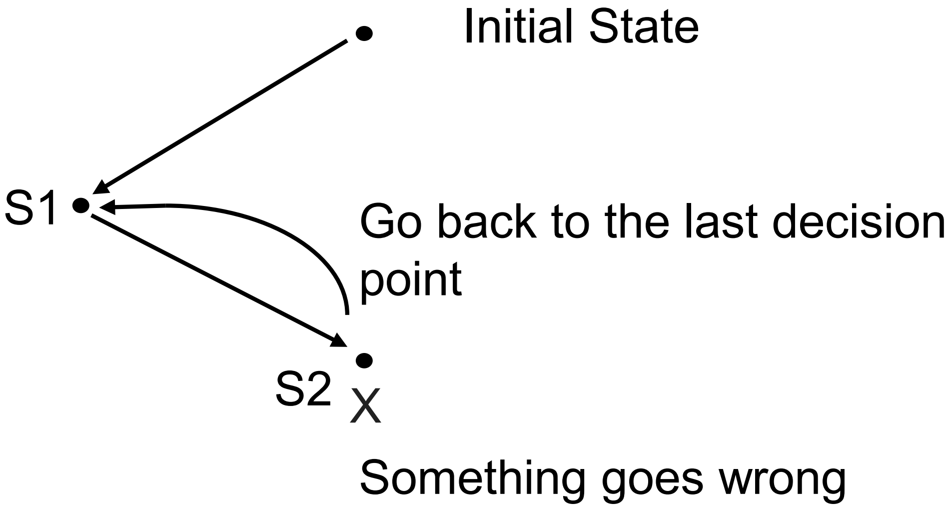
Example: map colouring



- Number of variables: n
- Max. depth of space: n
- Depth of solution state: n (all variables assigned)
- Search algorithm: depth-first search

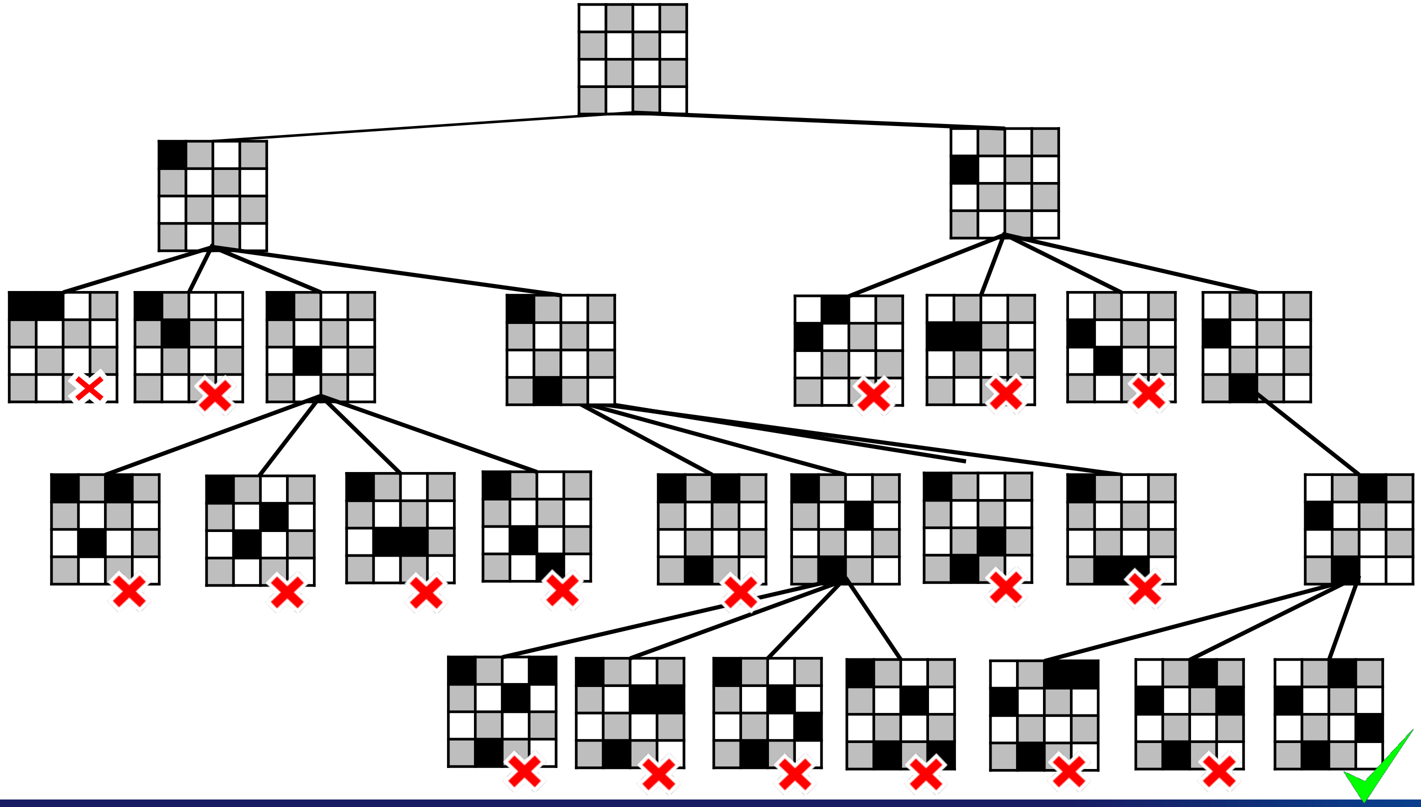
Backtracking Search

Backtracking search: Do not waste time searching when constraints have already been violated



- Before generating successors, check for constraint violations
- If yes, backtrack to try something else

Example (4-Queens)



Question to think:

- Please think about whether there are ways for further improving search efficiency.

Heuristics for CSPs

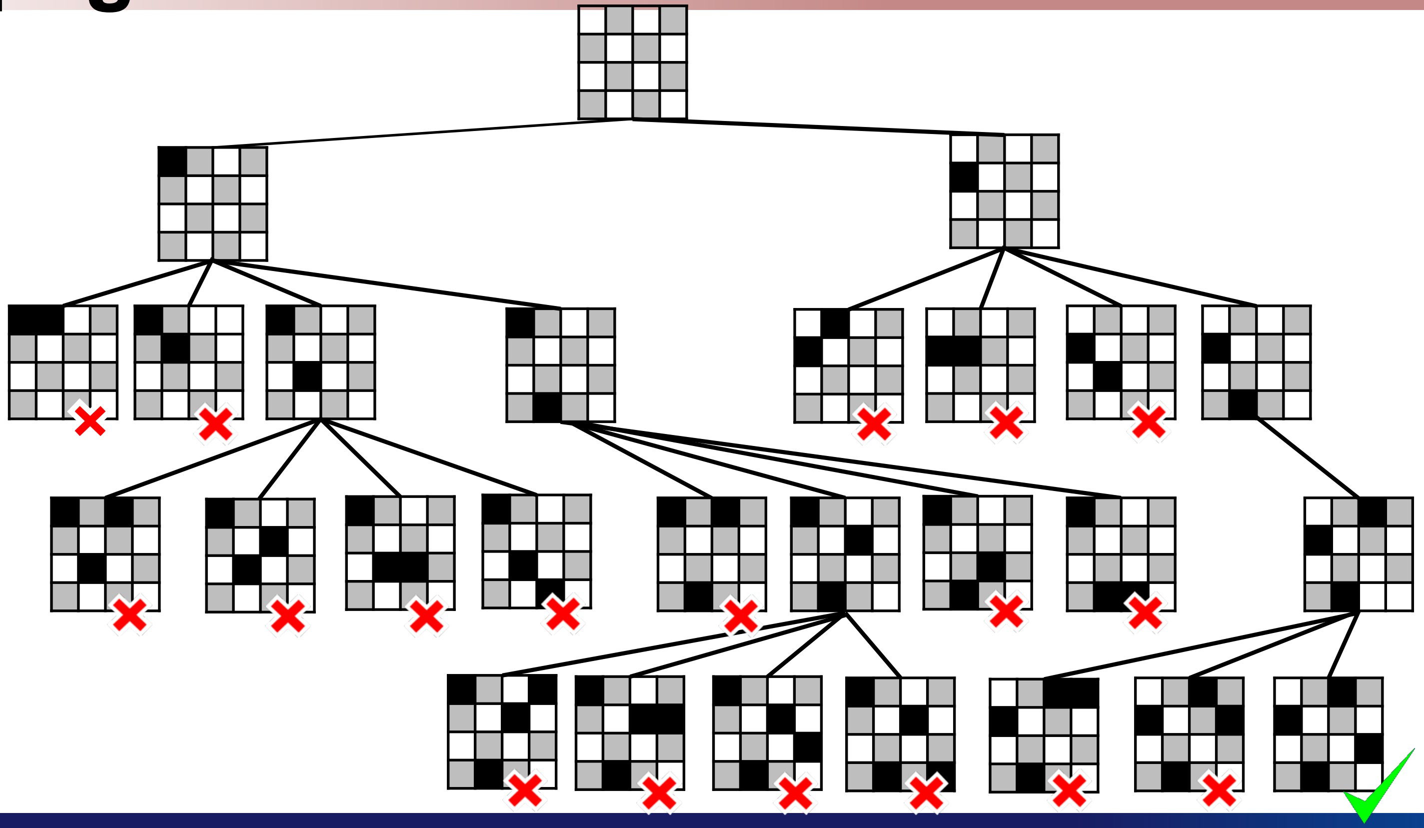
Plain backtracking is an uninformed algorithm!!

More intelligent search that takes into consideration

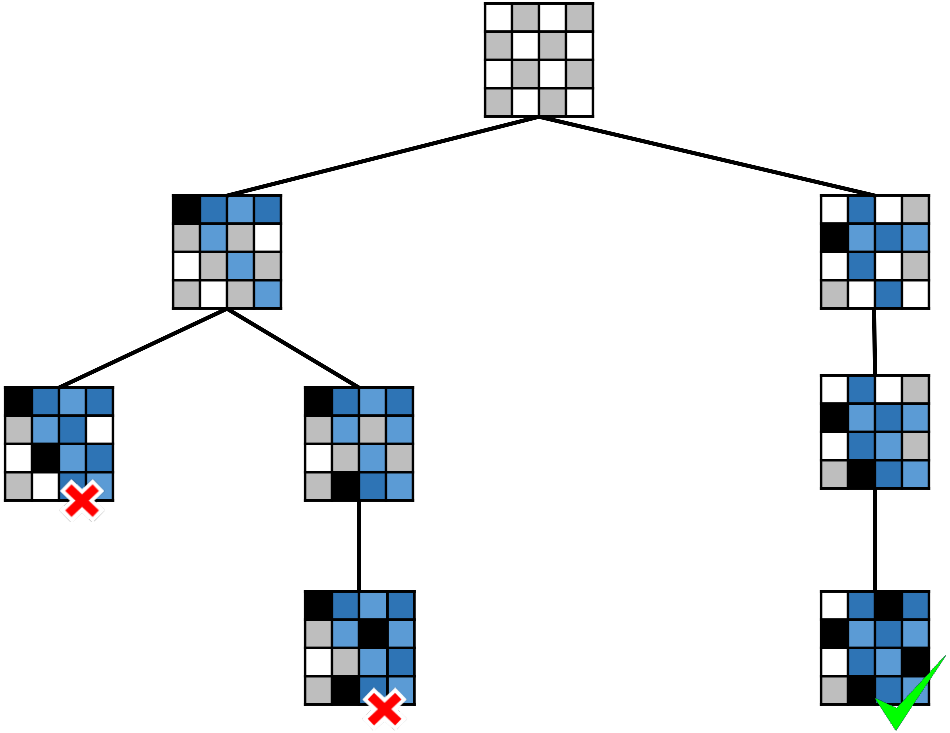
- Which variable to assign next
- What order of the values to try for each variable
- Implications of current variable assignments for the other unassigned variables
    - forward checking and **constraint propagation**

Constraint propagation: propagating the implications of a constraint on one variable onto other variables

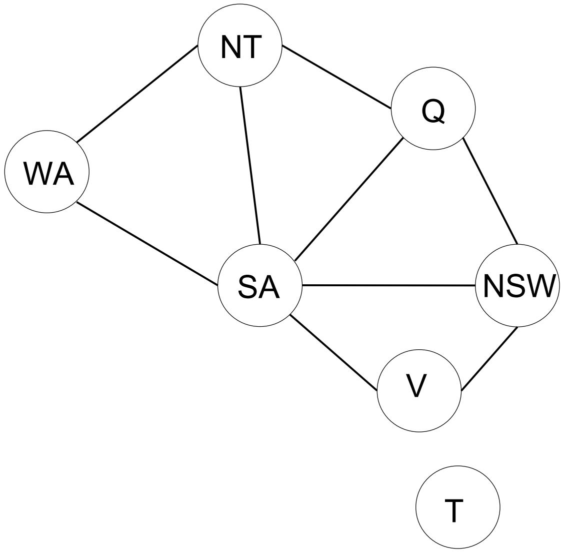
Example (4-Queens) without Constraint Propagation



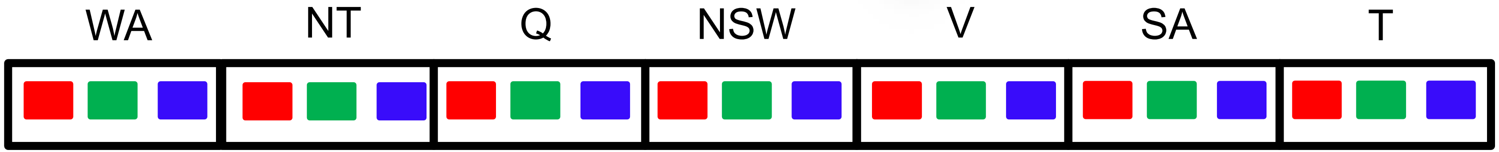
Search Tree of 4-Queens with Constraint Propagation



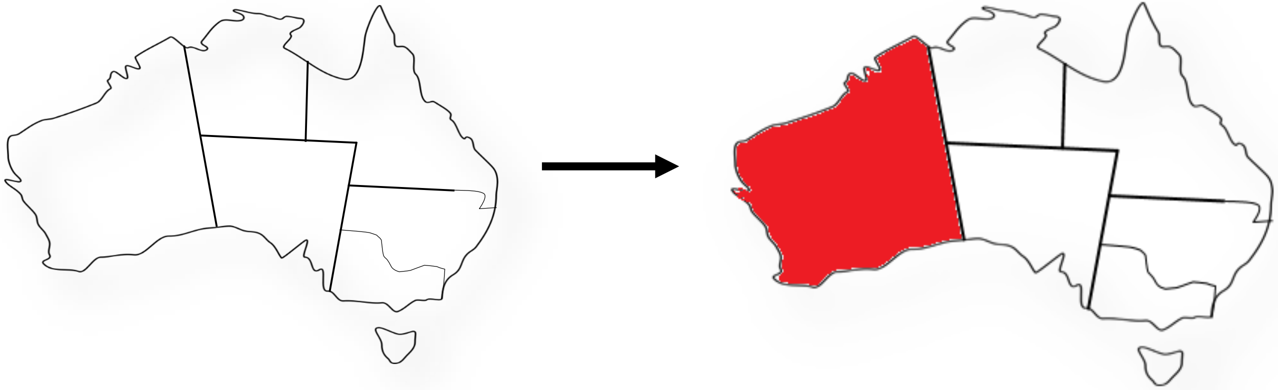
Example (Map Colouring)

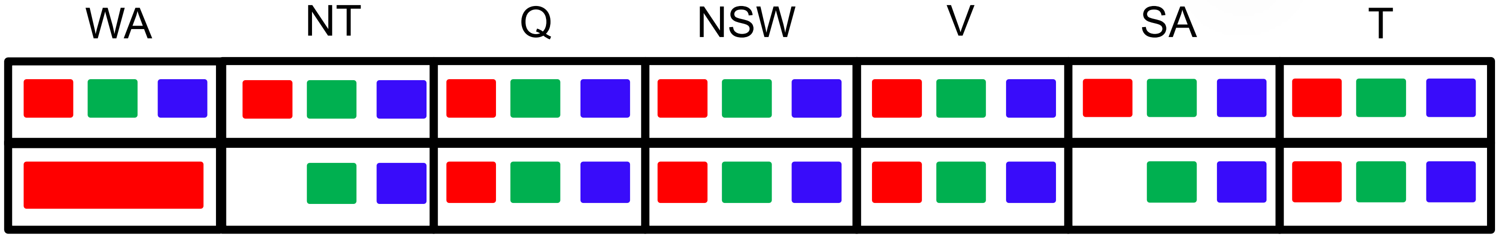


Example (Map Colouring)...

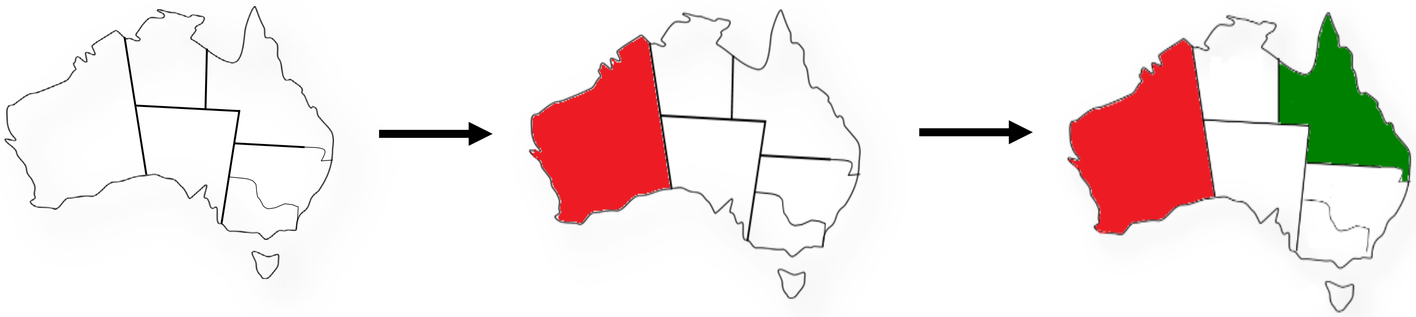


Example (Map Colouring)...

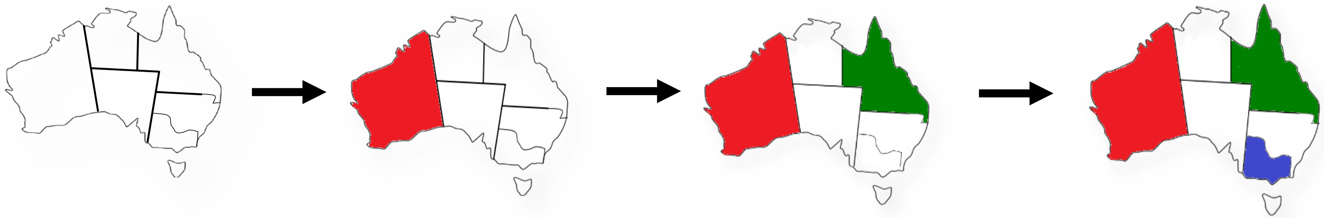




Example (Map Colouring)...



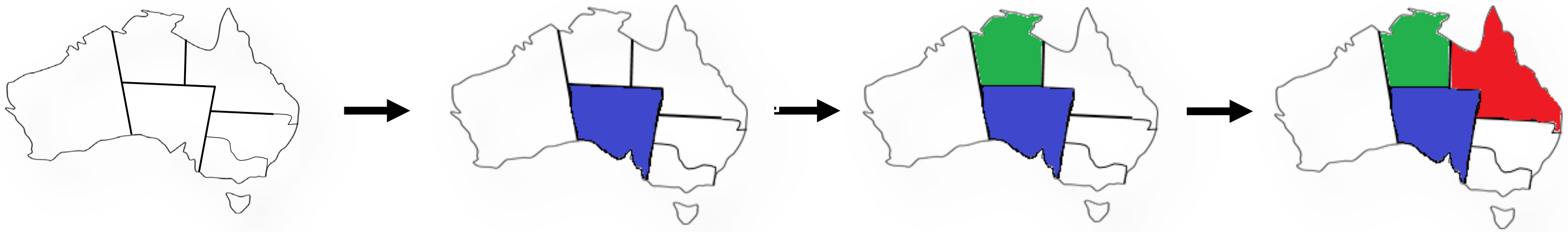
Example (Map Colouring)...



Most Constrained Variable

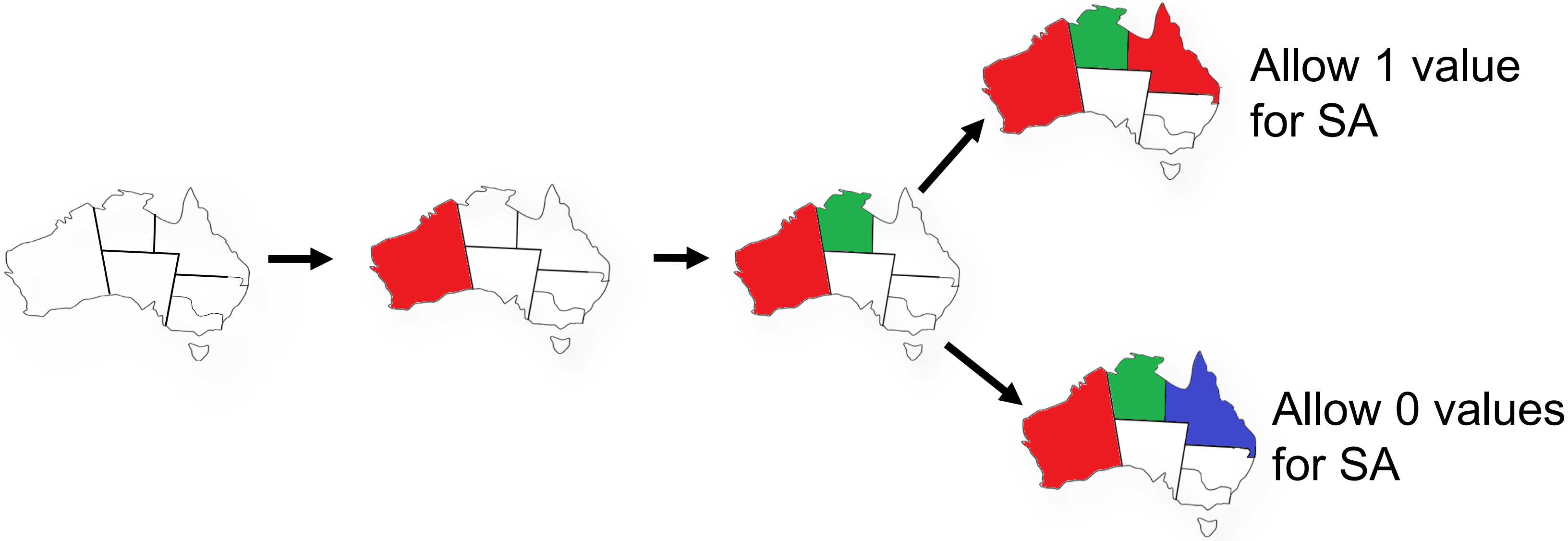
Or minimum remaining values (MRV) heuristic

Example: map colouring



To reduce the branching factor on future choices by selecting the variable that is involved in the **largest number of constraints** on unassigned variables.

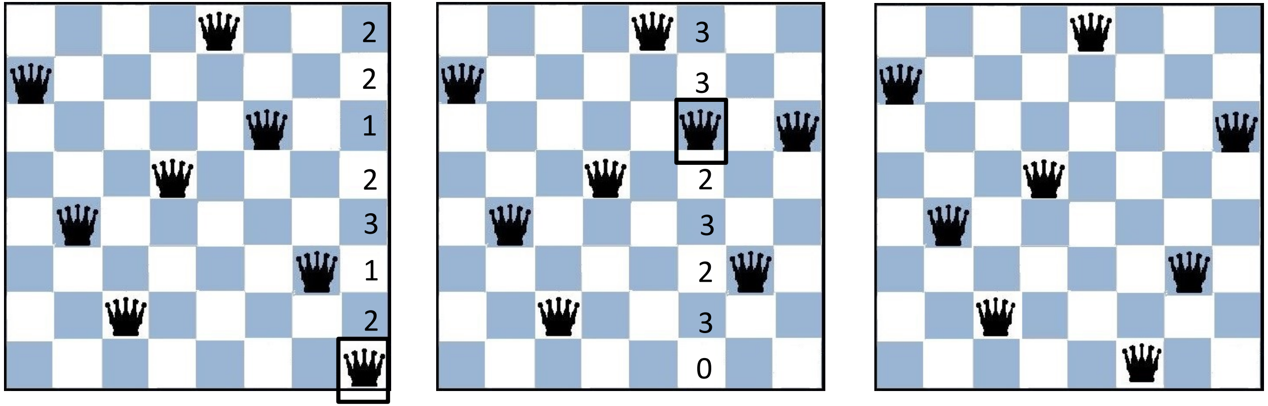
Least Constraining Value



Choose the value that leaves maximum flexibility for subsequent variable assignments

Min-Conflicts Heuristic (8-queens)

- A local heuristic search method for solving CSPs
- Given an initial assignment, selects a variable in the scope of a violated constraint and assigns it to the value that minimises the number of violated constraints



Question to think:

- Please think the intuitions for why the above ideas can improve search efficiency.

Games as Search Problems

Abstraction

- Ideal representation of real world problems
  - e.g. board games, chess, go, etc. as an abstraction of war games
  - Perfect information, i.e. fully observable
- Accurate formulation: state space representation

Uncertainty

- Account for the existence of hostile agents (players)
  - Other agents acting so as to diminish the agent's well-being
  - Uncertainty (about other agents' actions):
    - not due to the effect of non-deterministic actions
    - not due to randomness

    → Contingency problem

Games as Search Problems...

Complexity

- Games are abstract but not simple
  - e.g. chess: average branching factor = 35, game length > 50

  → complexity = 35^{50}(only 10^{40} for legal moves)

- Games are usually time limited
  - Complete search (for the optimal solution) not possible

  → uncertainty on actions desirability
  - Search efficiency is crucial

Types of Games

	Deterministic	Chance
Perfect information	Chess, Checkers, Go, Othello	Backgammon, Monopoly
Imperfect information		Bridge, Poker, Scrabble, Nuclear war

Perfect information

- each player has complete information about his opponent's position and about the choices available to him

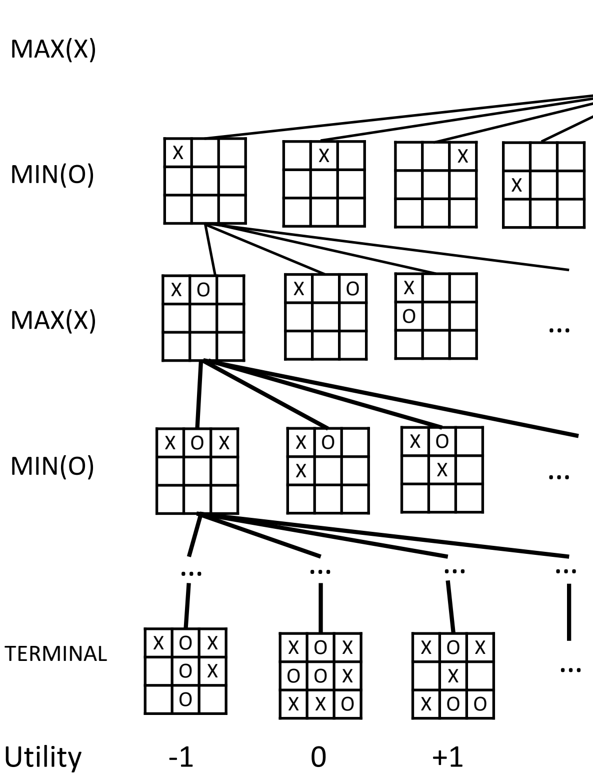
Game as a Search Problem

- Initial state: initial board configuration and indication of who makes the first move
- Operators: legal moves
- Terminal test: determines when the game is over
  - states where the game has ended: terminal states
- Utility function (payoff function): returns a numeric score to quantify the outcome of a game

Example: Chess

Win (+1), loss(-1) or draw (0)

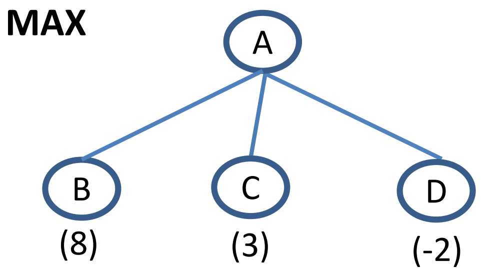
Game Tree for Tic-Tac-Toe



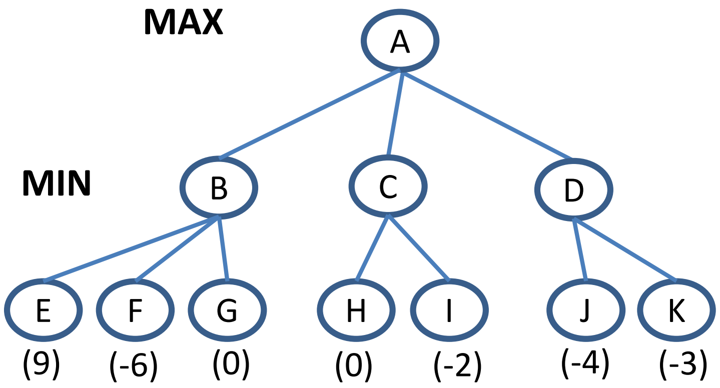
- The utility value of the terminal state is from the point of view of MAX
- MAX uses the search tree to determine the best move

What Search Strategy?

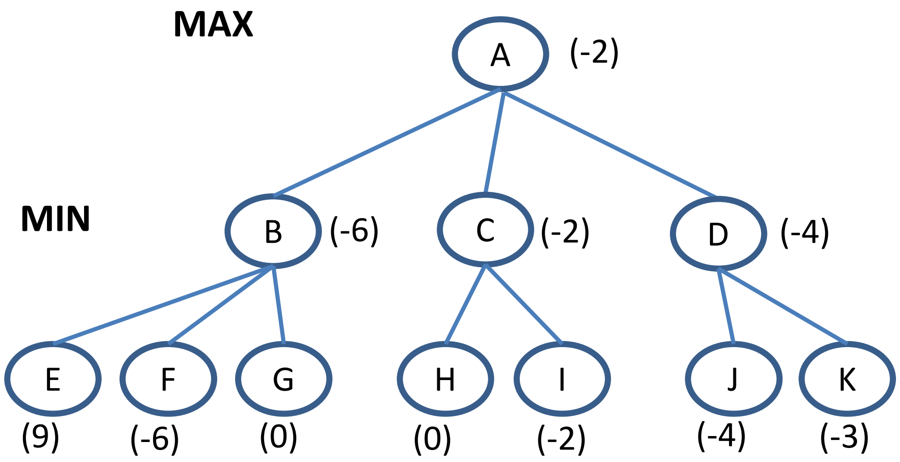
One-play



Two-play



What Search Strategy?...



Minimax Search Strategy

Search strategy

- Find a sequence of moves that leads to a terminal state (goal)

Minimax search strategy

- Maximise one's own utility and minimise the opponent's
    - Assumption is that the opponent does the same

Minimax Search Strategy

3-step process

1. Generate the entire game tree down to terminal states

2. Calculate utility
   a) Assess the utility of each terminal state
   b) Determine the best utility of the parents of the terminal state
   c) Repeat the process for their parents until the root is reached

3. Select the best move (i.e. the move with the highest utility value)

Perfect Decisions by Minimax Algorithm

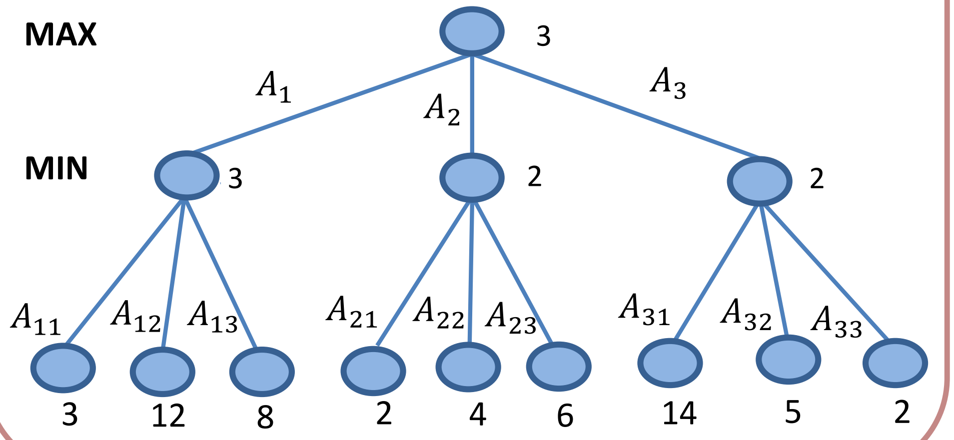
Perfect decisions: no time limit is imposed

- generate the complete search tree

Two players: MAX and MIN

- Choose move with best achievable payoff against best play
- MAX tries to max the utility, assuming that MIN will try to min it

Example



Othello 4

	X	O	
	O	X	

- A player can place a new piece in a position if there exists at least one straight (horizontal, vertical, or diagonal) occupied line between the new piece and another piece of the same kind, with one or more contiguous pieces from the opponent player between them
- After placing the new piece, the pieces from the opponent player will be captured and become the pieces from the same Player
- The player with the most pieces on the board wins

`X' plays first

X considers the game now

	X	O	
	O	X	

O considers the game now

	X	O	
	X	X	
	X		

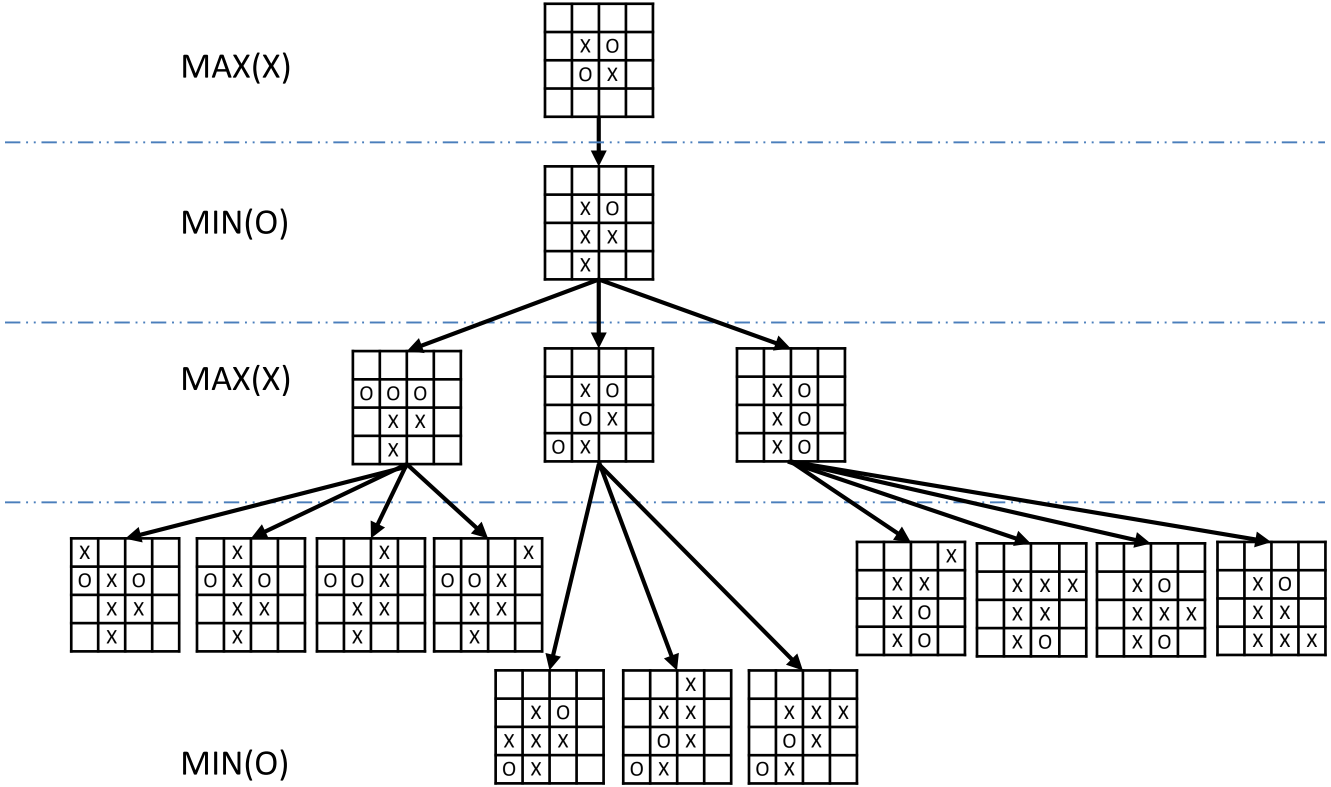
X considers the game now

O	O	O	
	X	X	
	X		

	X	O	
	O	X	
O	X		

	X	O	
	X	O	
	X	O	

Game Tree Othello 4



Imperfect Decisions

For chess, branching factor ≈ 35, each player typically makes 50 moves → for the complete game tree, need to examine 35^{100} positions

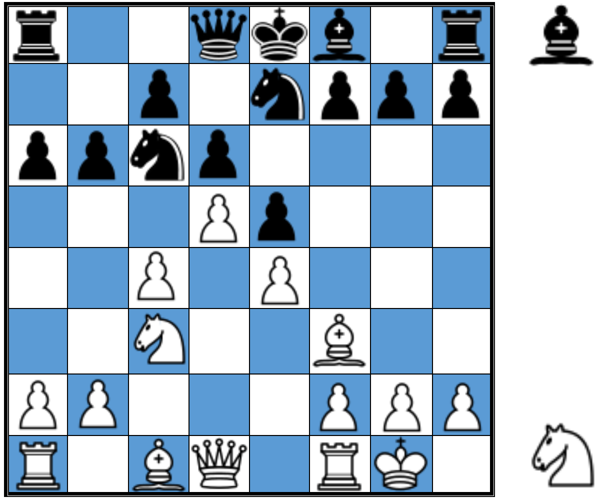
Time/space requirements → complete game tree search is intractable → impractical to make perfect decisions

Modifications to minimax algorithm

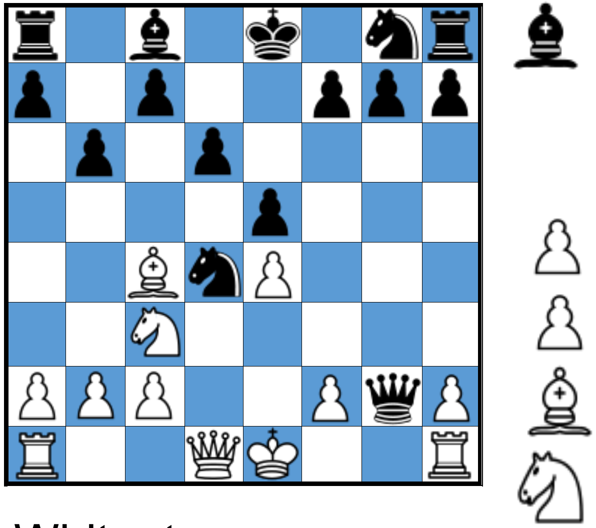
1. replace utility function by an estimated desirability of the position
   - Evaluation function
2. partial tree search
   - E.g., depth limit
   - Replace terminal test by a cut-off test

Evaluation Functions

Returns an estimate of the expected utility of the game from a given position



Black: to move
White: slightly better



White: to move
Black: winning