Constraint Satisfaction Problems

Part III

Constraint Satisfaction Problems (CSPs) • CSP examples

Backtracking search and heuristics • Forward checking

Arc consistency • Variable elimination • Local Search

UNSW SYDNEY

Constraint Satisfaction Problems (CSPs)

- Constraint Satisfaction Problems are defined by a set of variables X_i, each with a domain D_i of possible values, and a set of constraints C that specify allowable combinations of values.

- The aim is to find an assignment of the variables X_i from the domains D_i in such a way that none of the constraints C are violated.
  - i.e. all of the constraints C are satisfied

Example: Map-Colouring

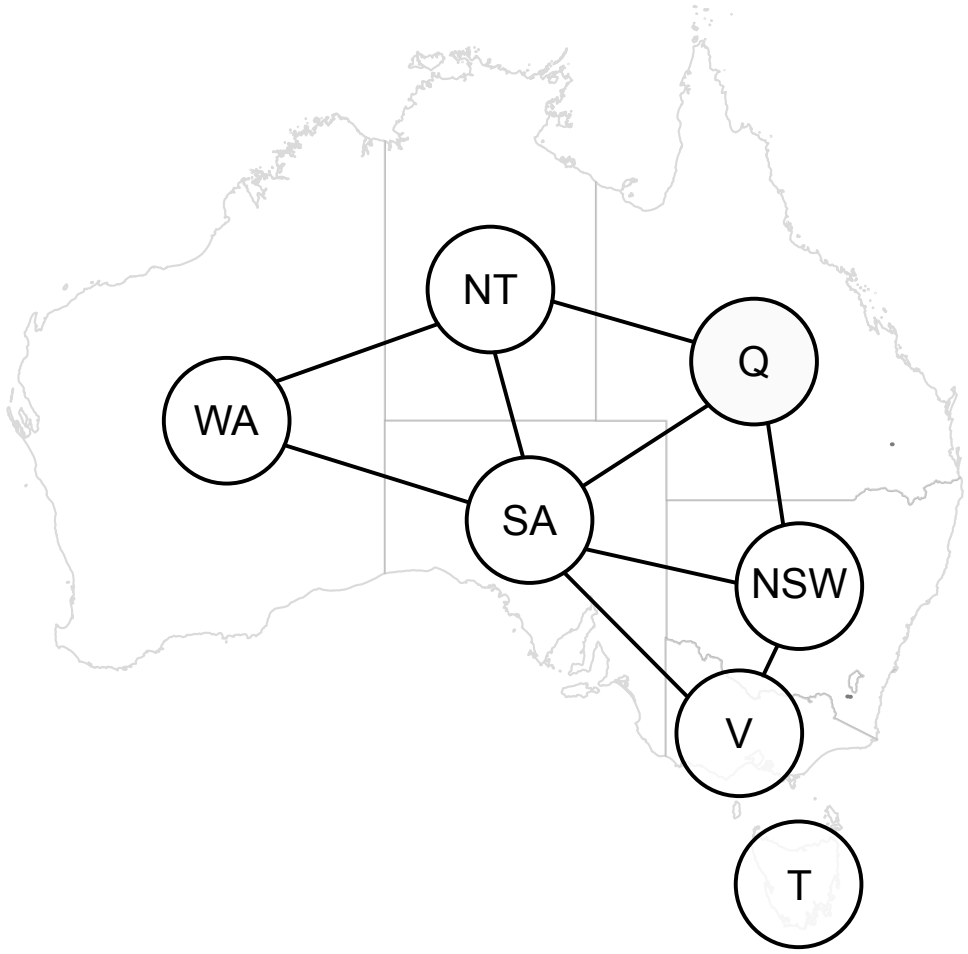
Variables: WA, NT, Q, NSW, V, SA, T

Domains: Di = {red, green, blue}

Constraints: adjacent regions must have different colours **e.g. WA≠ NT, etc.**

Constraint graph

- Constraint graph: nodes are variables, arcs are constraints
- Binary CSP: each constraint relates two variables



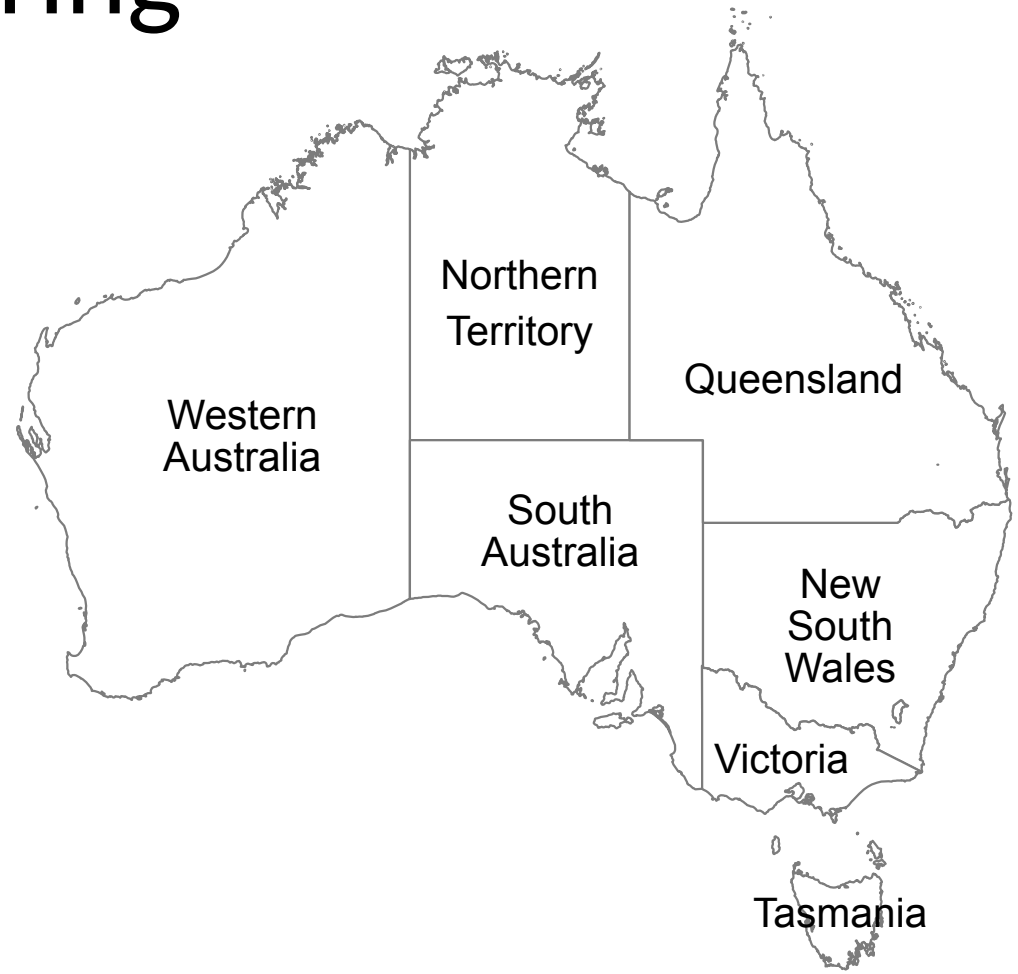
Example: Map-Colouring

Variables: WA, NT, Q, NSW, V, SA, T

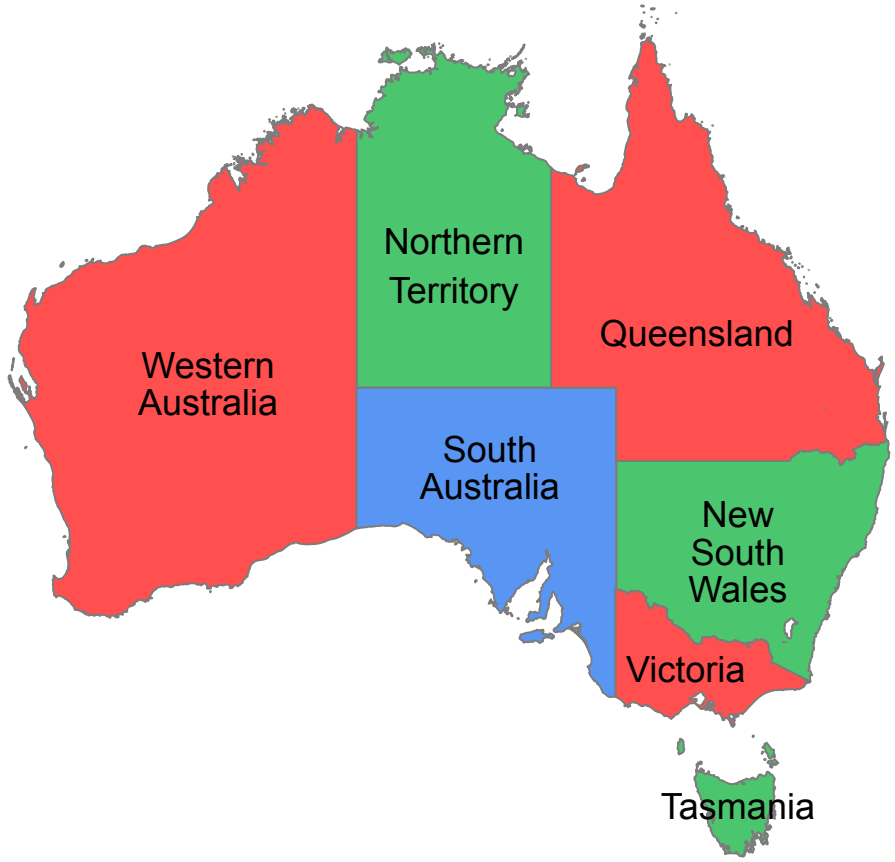
Domains: Di = {red, green, blue}

Constraints: adjacent regions must have different colours
e.g. WA≠ NT, etc.

or (WA,NT) in {(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)}



Example: Map-Colouring

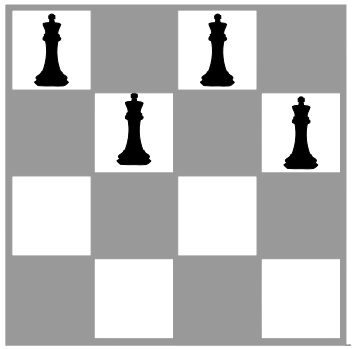


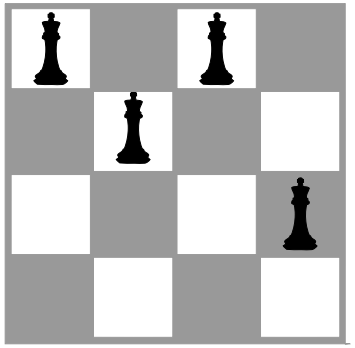
{WA=red, NT=green, Q=red, NSW=green, V=red, SA=blue, T=green}

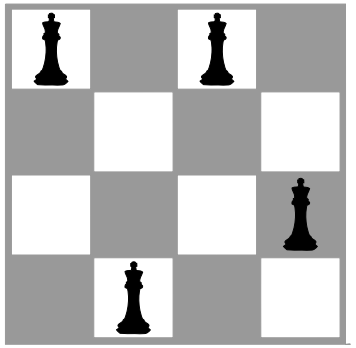
n-Queens Puzzle as a CSP

Assume one queen in each column. Domains are possible positions of queen in a column. Assignment is when each domain has one element.

Which row does each one go in?







Variables: $Q1$, $Q2$, $Q3$, $Q4$

Domains: Di ={1, 2, 3, 4}

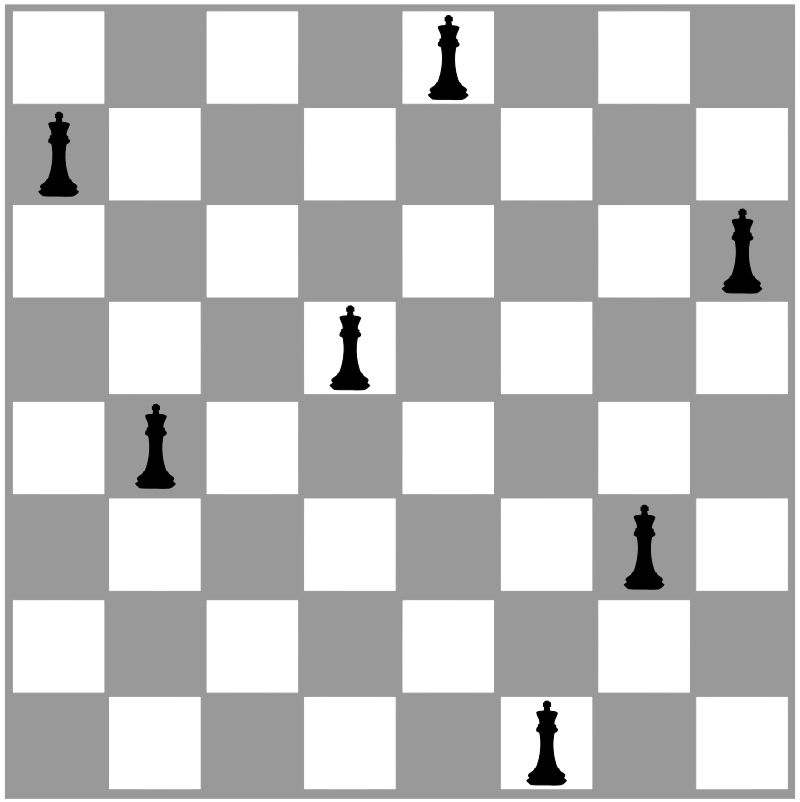
Constraints:

$Qi \neq Qj$ (cannot be in same row)

$|Qi - Qj| \neq |i - j|$ (or same diagonal)

Example: n-Queens Puzzle

Put n queens on an n-by-n chess board so that no two queens are attacking each other.



Example: Cryptarithmetic

Variables:
D E M N O R S Y

Domains:
{0,1,2,3,4,5,6,7,8,9}

Constraints:
$M \neq 0$, $S \neq 0$ (unary constraints)
Y = D+E or Y = D+E −10, etc.
D ≠ E, D ≠ M, D ≠ N, etc.

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

Example: Cryptarithmetic

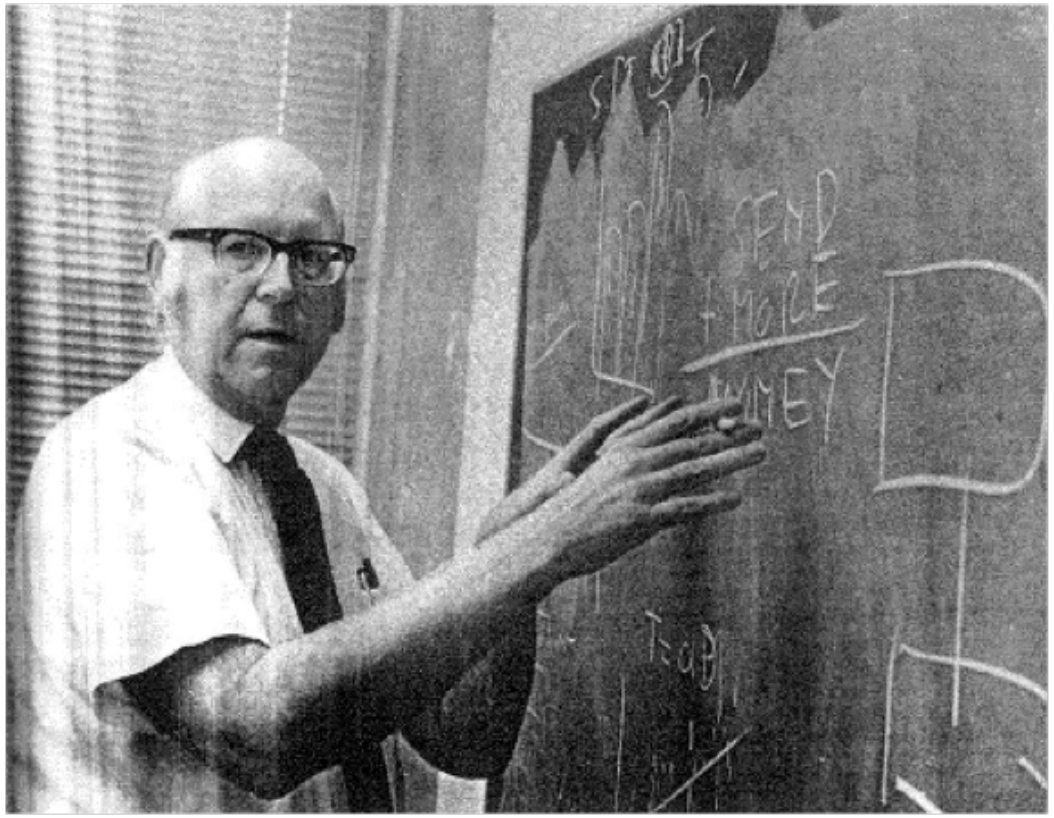
Variables: F T U W R O X_1 X_2 X_3

Domains: {0,1,2,3,4,5,6,7,8,9}

Constraints: AllDifferent(F,T,U,W,R,O)
O + O = R + 10·X_1, etc.

```
    T W O
+
    T W O
-----------
  F O U R
```

Cryptarithmetic with Allen Newell



Book: Intended Rational Behavior

Cryptarithmetic with Allen Newell

7.1. Cryptarithmetic

Let us start with cryptarithmetic. This task was first analyzed by Bartlett (1958) and later by Herb Simon and myself (Newell & Simon, 1972). It plays an important role in the emergence of cognitive psychology—at least for me, and perhaps for others. It has been the strongest convincer that humans really do use problem spaces and do search in them, just as the AI theory of heuristic search says.

A cryptarithmetic task is just a small arithmetical puzzle (see Figure 7-1). The words *DONALD*, *GERALD*, and *ROBERT* represent three six-digit numbers. Each letter is to be replaced by a distinct digit (that is, D and T must each be a digit, say $D = 5$ and $T = 0$, but they cannot be the same digit). This replacement must lead to a correct sum, such that *DONALD* + *GERALD* = *ROBERT*. Mathematically viewed, the problem is one of satisfying multiple integer constraints involving equality, inequality, and unequality.

Humans can be set to solving cryptarithmetic tasks, and pro-

Cryptarithmetic with Allen Newell

Assign each letter a unique digit to make a correct sum

```
  DONALD
+ GERALD      D = 5
--------
 ROBERT
```

Figure 7-1. The cryptarithmetic task.

tocols can be obtained from transcripts of their verbalizations while they work (Newell & Simon, 1972). Analysis shows that people solve the task by searching in a problem space and that the search can be plotted explicitly.[1] Figure 7-2 shows the behavior of S3 on *DONALD + GERALD = ROBERT*.[2] The plot is called a *problem-behavior graph (PBG)*; it is just a way of spreading out the search so it can be examined (Figure 1-4 showed a PBG for chess). S3

Cryptarithmetic with Allen Newell

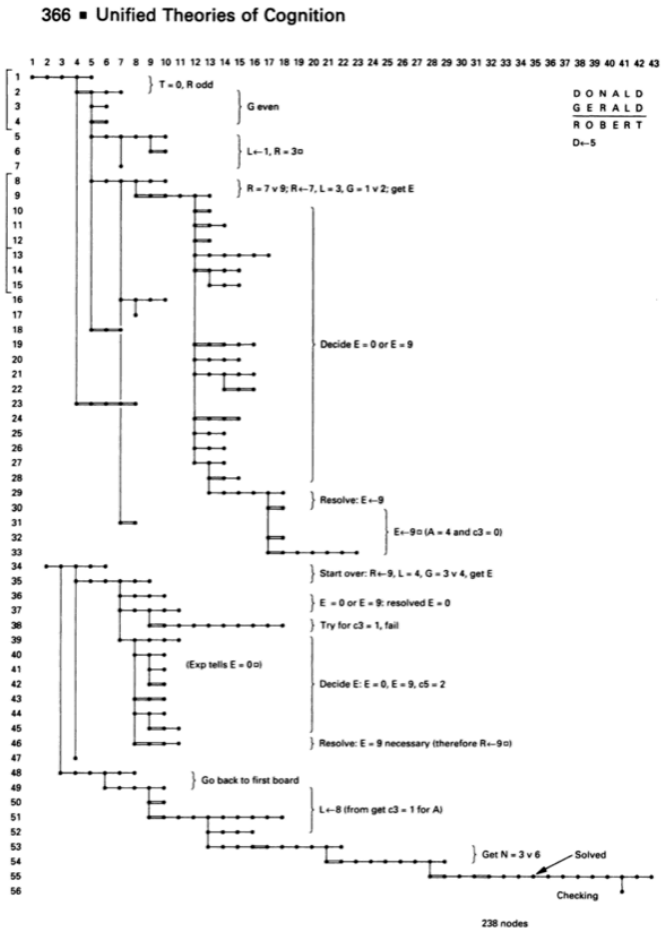
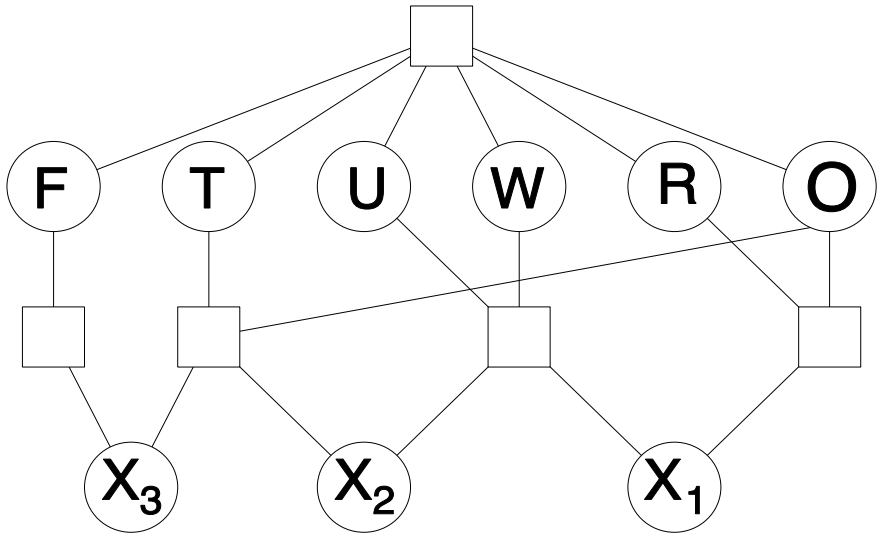


Figure 7-2. Problem-behavior graph of subject S3 on *DONALD + GERALD = ROBERT.*

Cryptarithmetic with Hidden Variables

- We can add “hidden” variables to simplify the constraints.

```
    T W O
+
    T W O
-----------
  F O U R
```



Variables: F T U W R O X_1 X_2 X_3
Domains: {0,1,2,3,4,5,6,7,8,9}

Constraints:
AllDifferent(F,T,U,W,R,O)
$O + O = R + 10 \cdot X_1$, etc.

Real-world CSPs

- Assignment problems (e.g. who teaches what class)
- Timetabling problems (e.g. which class is offered when and where?)
- Hardware configuration (e.g. minimise space for circuit layout)
- Transport scheduling (e.g. courier delivery, vehicle routing)
- Factory scheduling (optimise assignment of jobs to machines)
- Gate assignment (assign gates to aircraft to minimise transit)

Many real world CSPs are also optimisation problems

Real-world CSPs - Factory scheduling

- A robot (agent) needs to schedule a set of activities for a manufacturing process, involving casting, milling, drilling, and bolting.
- Each activity has a set of possible times at which it may start.
- The robot has to satisfy various constraints arising from prerequisite requirements and resource use limitations.
- For each activity there is a variable that represents the time that it starts:
  - B – start of bolding
  - D – start of drilling
  - C – start of casting

Real-world CSPs - Factory scheduling

- A robot (agent) needs to schedule a set of activities for a manufacturing process, involving casting, milling, drilling, and bolting.
- Each activity has a set of possible times at which it may start.
- The robot has to satisfy various constraints arising from prerequisite requirements and resource use limitations.
- For each activity there is a variable that represents the time that it starts:
  - B – start of bolding
  - D – start of drilling
  - C – start of casting
- Constraints: $D < B$, $C \neq D$, $B = C+3$

Real-world CSPs - Factory scheduling

- Consider a constraint on the possible dates for three activities.

$$(A \leq B) \wedge (B < 3) \wedge (B < C) \wedge \neg (A = B \wedge C \leq 3)$$

Variables: A, B, C - variables that represent the date of each activity
Domain of each variable is: {1, 2, 3, }
A **constraint** with scope:
This formula says that A is on the same date or before B
and it cannot be that A and B are on the same date and C is on or before day 3.

Real-world CSPs - Factory scheduling

- Consider a constraint on the possible dates for three activities.

Variables: A, B, C - variables that represent the date of each activity
Domain of each variable is: {1, 2, 3, }
A constraint with scope:

$(A \leq B) \wedge (B < 3) \wedge (B < C) \wedge \neg (A = B \wedge C \leq 3)$

This formula says that A is on the same date or before B
and it cannot be that A and B are on the same date and C is on or before day 3.

This constraint could instead have its relation defined its extension, as a table specifying the legal assignments:

A	B	C
2	2	4
1	1	4
1	2	3
1	2	4

Types of constraints

- Unary constraints involve a single variable
  - M ≠ 0
- Binary constraints involve pairs of variables
  - SA ≠ WA
- Higher-order constraints involve 3 or more variables
  - $Y = D + E$ or $Y = D + E - 10$
- Inequality constraints on Continuous variables
  - EndJob1 + 5 ≤ StartJob3
- Soft constraints (Preferences)
  - 11am lecture is better than 8am lecture!

Path Search vs Constraint Satisfaction

Important difference between path search problems and CSPs

- Path Search Problems (e.g. Rubik's Cube)
  - Knowing the final state is easy
  - Difficult part is how to get there

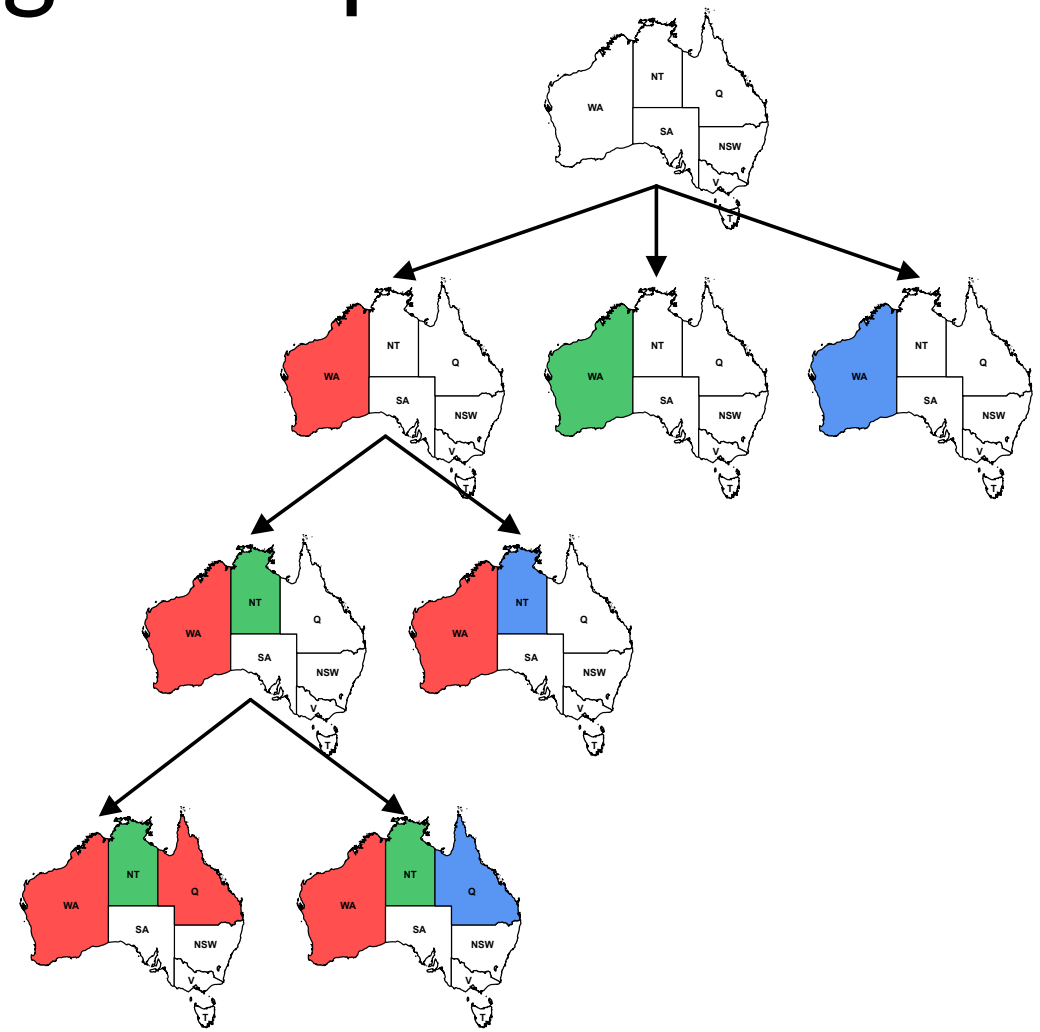
- Constraint Satisfaction Problems (e.g. n-Queens)
  - Difficult part is knowing the final state
  - How to get there is easy

Backtracking Search

CSPs can be solved by assigning values to variables one by one, in different combinations. Whenever a constraint is violated, we go back to the most recently assigned variable and assign it a new value.
This can be achieved by a Depth First Search on a special kind of state space, where states are defined by the values assigned so far:

- Initial state: the empty assignment.
- Successor function: assign a value to an unassigned variable that does not conflict with previously assigned values of other variables. (If no legal values remain, the successor function fails.)
- Goal test: all variables have been assigned a value, and no constraints have been violated.

Backtracking example



Path Search vs. Constraint Satisfaction

Important difference between Path Search Problems and CSP's:

- Constraint Satisfaction Problems (e.g. n-Queens)
  - difficult part is knowing the final state
  - how to get there is easy
- Path Search Problems (e.g. Rubik's Cube)
  - knowing the final state is easy
  - difficult part is how to get there

Backtracking search Space Properties

The search space for this Depth First Search has certain very specific properties:

- If there are n variables, every solution will occur at exactly depth n.
- Variable assignments are commutative

  [WA = red then NT = green] same as [NT = green then WA = red]

Backtracking search can solve n-Queens for $n \approx 25$

Improvements to Backtracking search

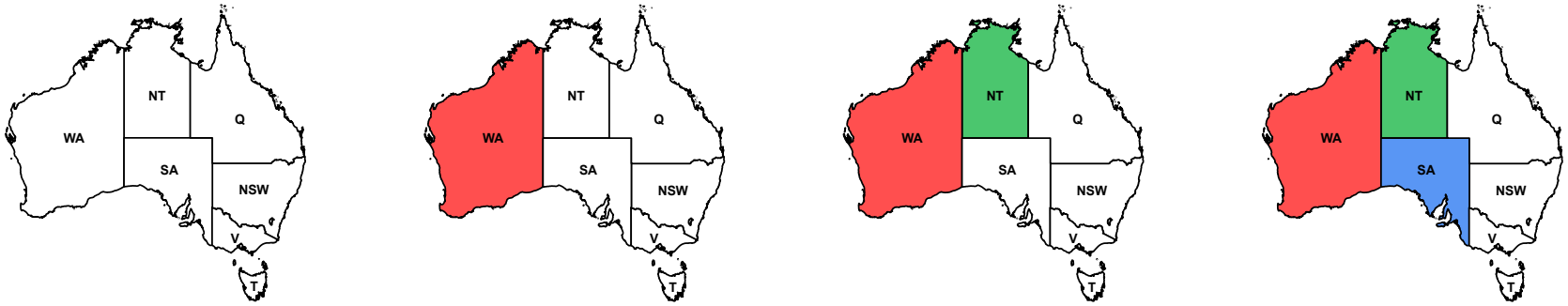
General-purpose methods can give huge gains in speed:

1. Which variable should be assigned next?
2. In what order should its values be tried?
3. Can we detect inevitable failure early?

Improving backtracking efficiency

Minimum Remaining Values

- Minimum Remaining Values (MRV)
  - choose the variable with the fewest legal values

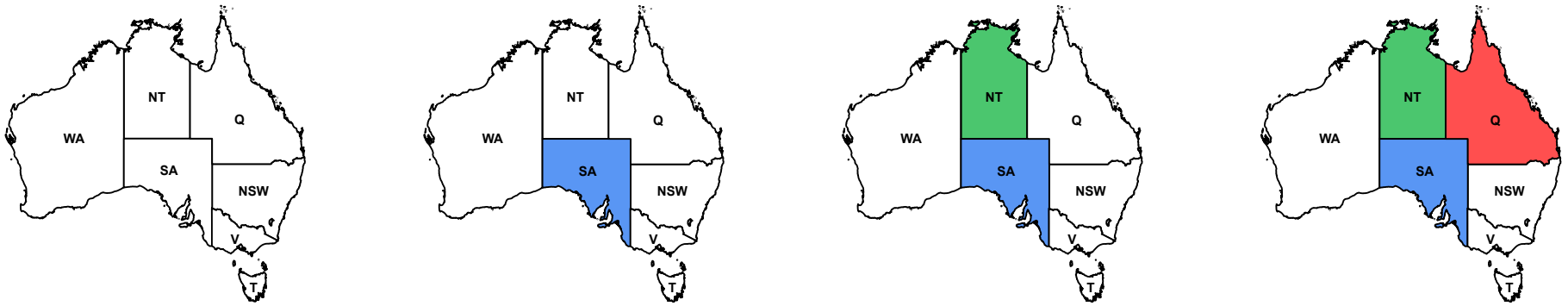


  - Most constrained variable

- Minimum remaining values (MRV) heuristic

Degree Heuristic

- Tie-breaker among MVR variables
- Degree heuristic:
  - choose the variable with the most constraints on remaining variables



Least Constraining Value

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

Allows 2 values for NT, SA

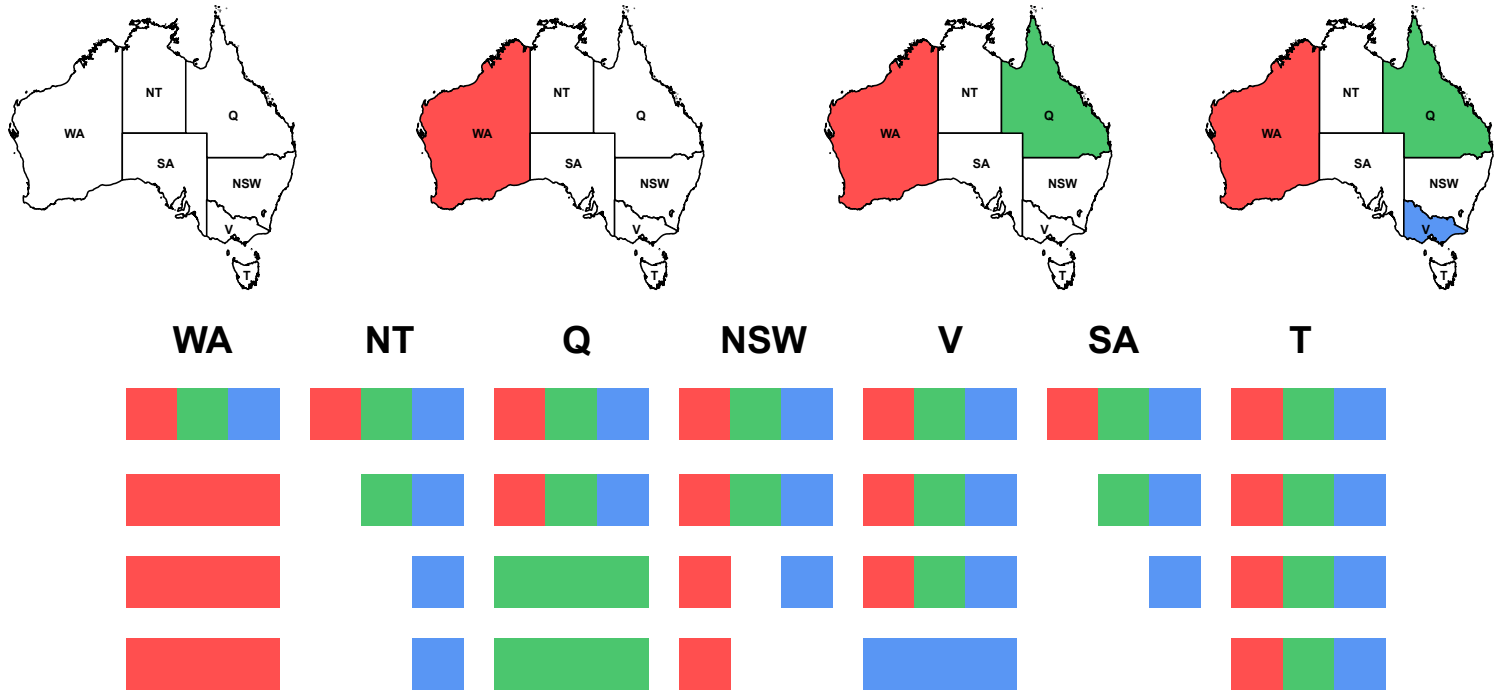
Allows 1 values for NT, SA

- (More generally, 3 allowed values would be better than 2, etc.)
  - Combining these heuristics makes 1000 queens feasible.

Forward checking

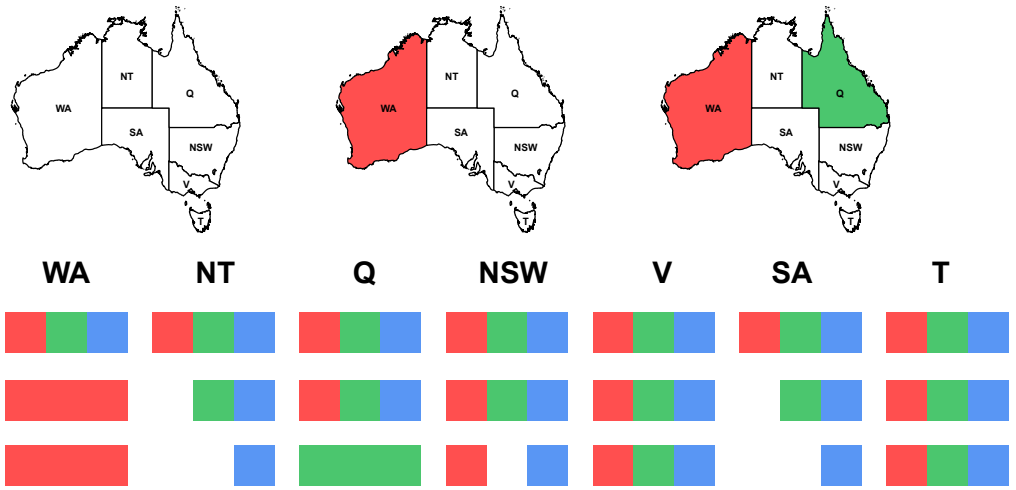
Idea:

- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values
    - prune off that part of the search tree, and backtrack



Constraint propagation

Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:



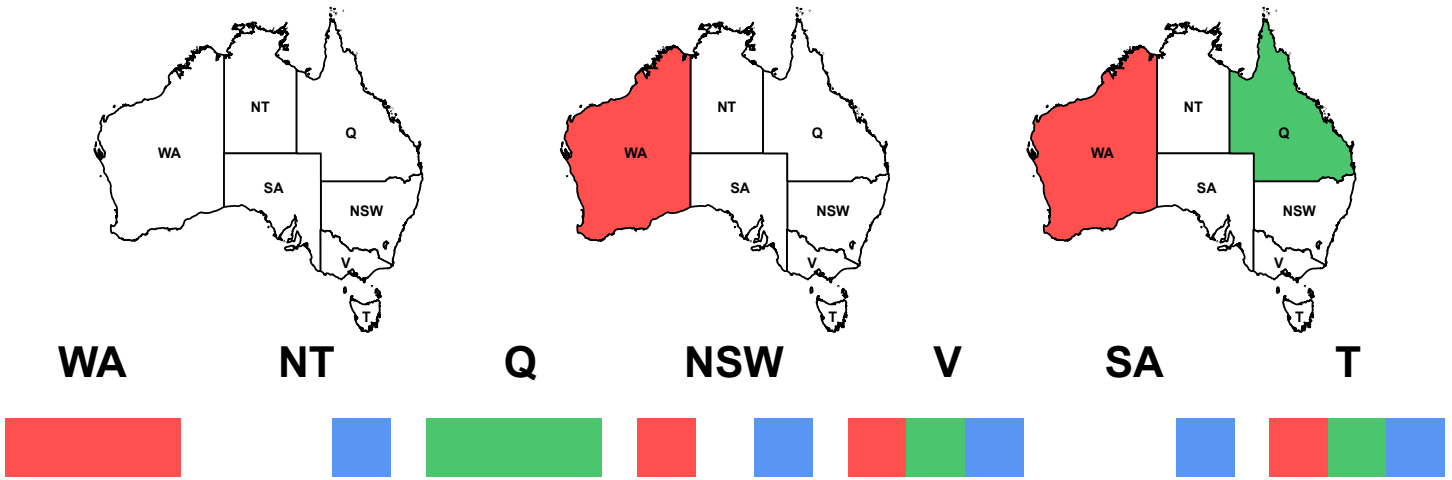
NT and SA cannot both be blue!

Constraint propagation repeatedly enforces constraints locally

Arc consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent if *for* ***every*** *value x of X there is* ***some*** *allowed y*

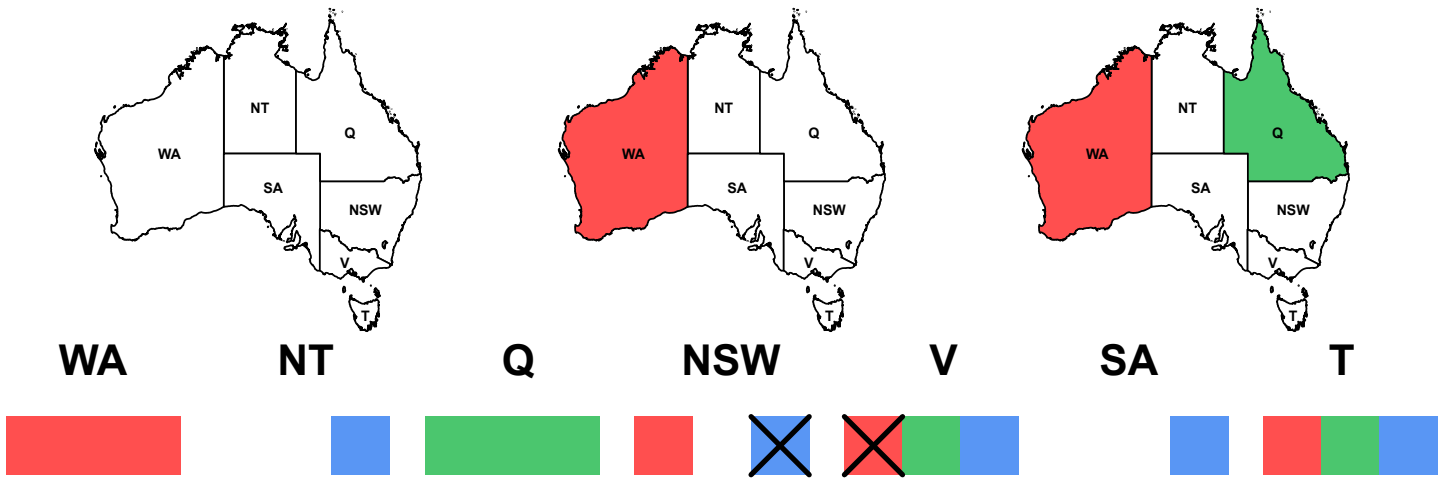


If X loses a value, neighbours of X need to be rechecked.

Arc consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent if *for* ***every*** *value x of X there is* ***some*** *allowed y*



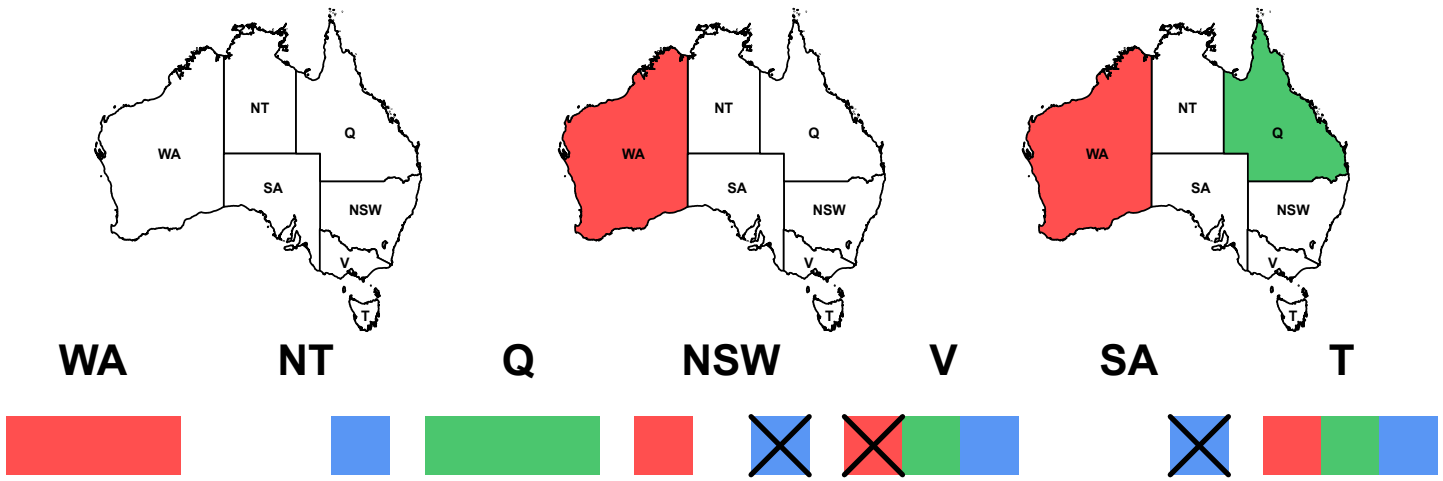
Arc consistency detects failure earlier than forward checking

Can be run as a preprocessor after each assignment

Arc consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent if *for* ***every*** *value x of X there is* ***some*** *allowed y*

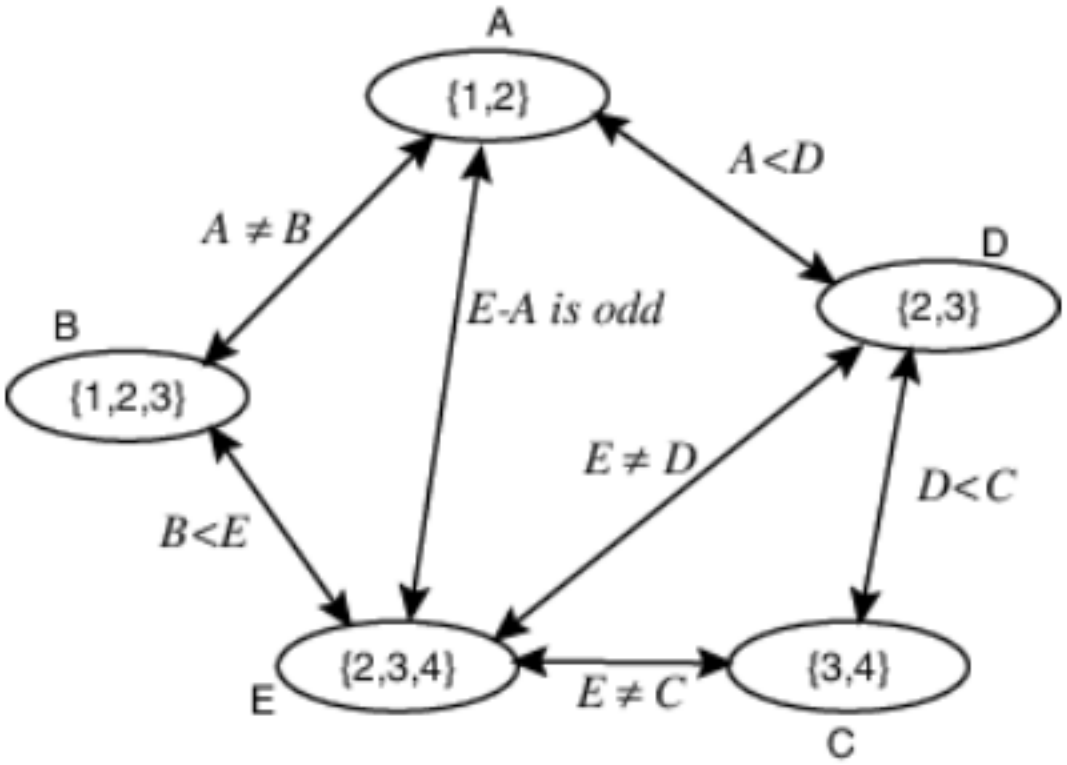


For some problems, it can speed up the search enormously.
For others, it may slow the search due to computational overheads.

Variable Elimination

- If there is only one variable, return the intersection of its (unary) constraints
- Otherwise
  - Select a variable X
  - Join the constraints in which X appears, forming constraint R1
  - Project R1 onto its variables other than X, forming R2
  - Replace all of the constraints in which X appears by R2
  - Recursively solve the simplified problem, forming R3
  - Return R1 joined with R3

Variable Elimination

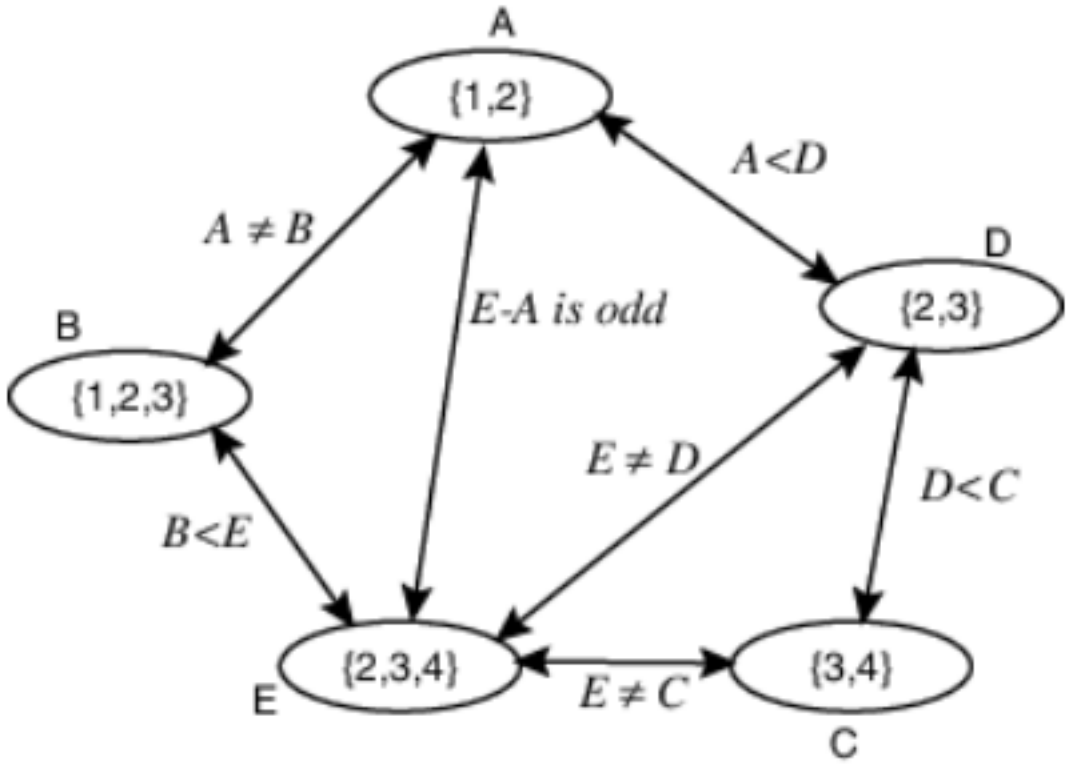


Variable Elimination

Constraints:
A ≠ B, E ≠ C, E ≠ D, A < D, B < E,
D < C, E-A is odd

Variables: A, B, C, D, E

Domains: A = {1,2}, B = {1,2,3}, C = {3,4}, D = {2,3}, E ={2,3,4}



Variable Elimination Example

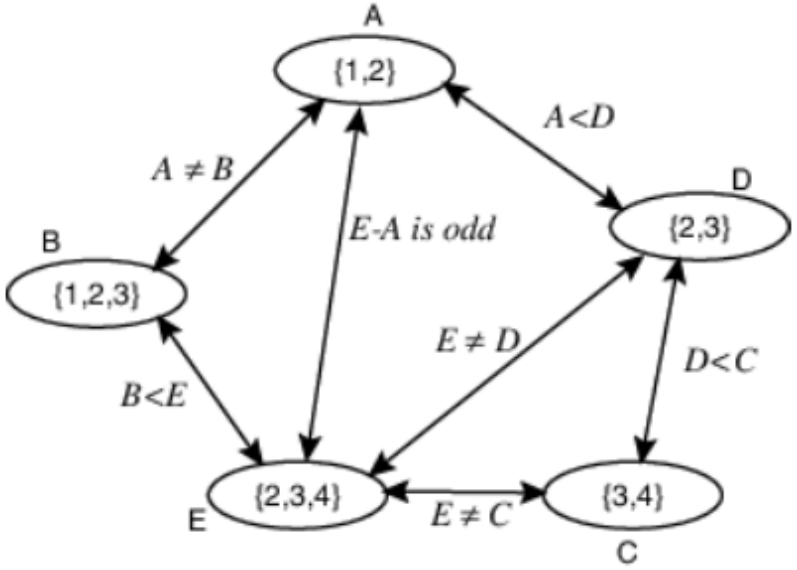
$r_1 : C \neq E$	C	E
	3	2
	3	4
	4	2
	4	3

$r_2 : C > D$	C
	3
	4
	4

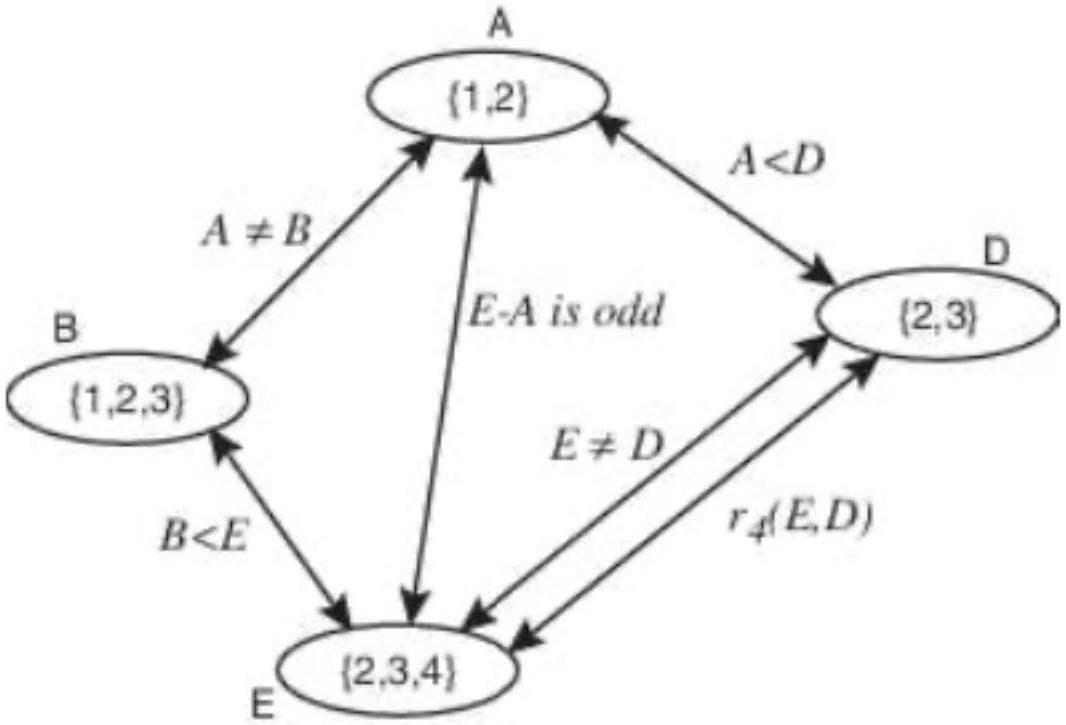
$r_3 : r_1 \bowtie r2$	C	D	E
	3	2	2
	3	2	4
	4	2	2
	4	2	3
	4	3	2
	4	3	3

$r_4 : \pi_{\{D,E\}} r_3$	D	E
	2	2
	2	3
	2	4
	3	2
	3	3

↪ new constraint

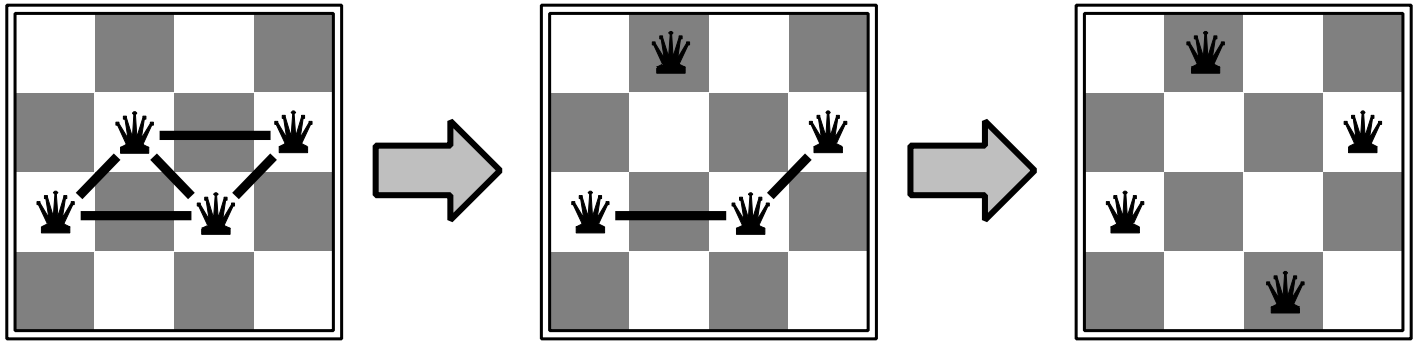


Variable Elimination Example



Local Search

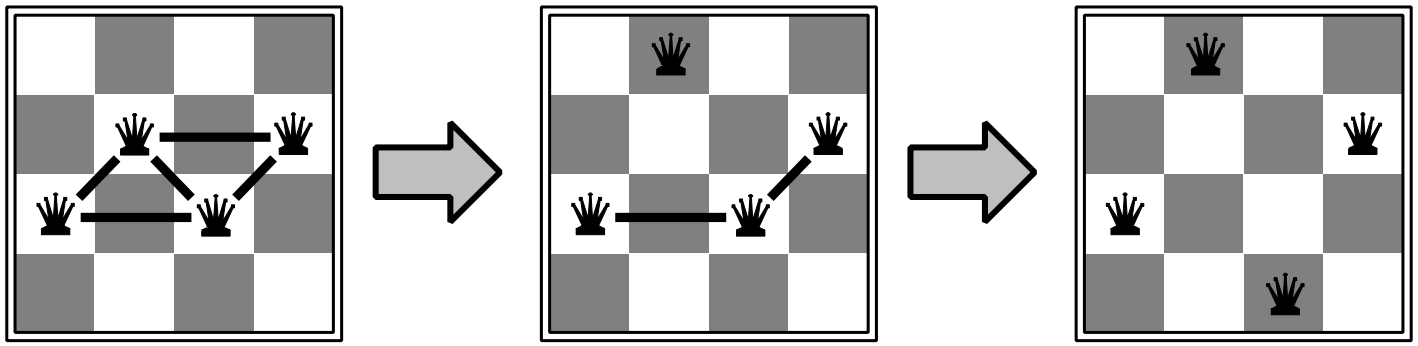
There is another class of algorithms for solving CSP's, called "Iterative Improvement" or "Local Search".



Local Search

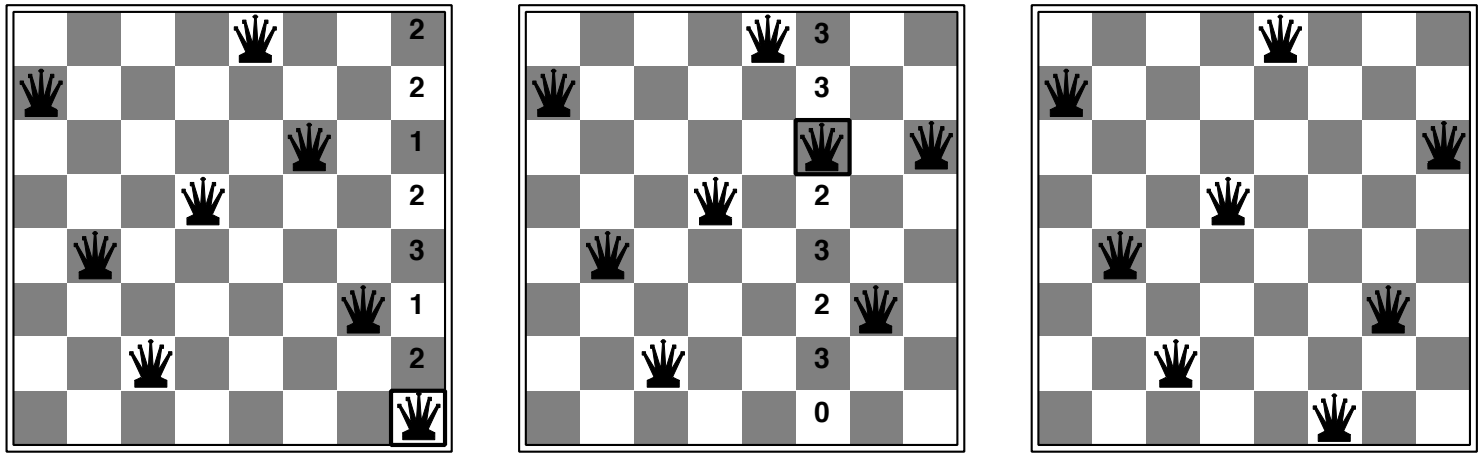
There is another class of algorithms for solving CSP's, called "Iterative Improvement" or "Local Search".

- Iterative Improvement
  - assign all variables randomly in the beginning (thus violating several constraints),
  - change one variable at a time, trying to reduce the number of violations at each step.
  - Greedy Search with h = number of constraints violated



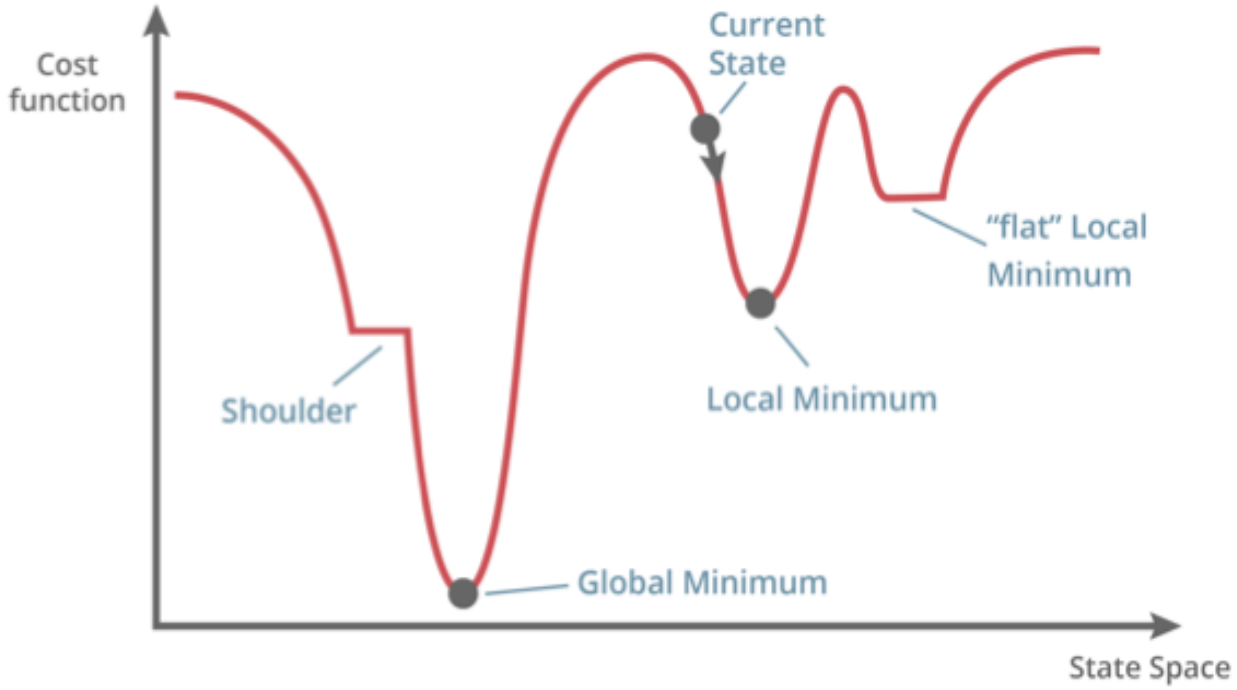
h = 5 **h = 2** **h = 0**

Hill-climbing by min-conflicts



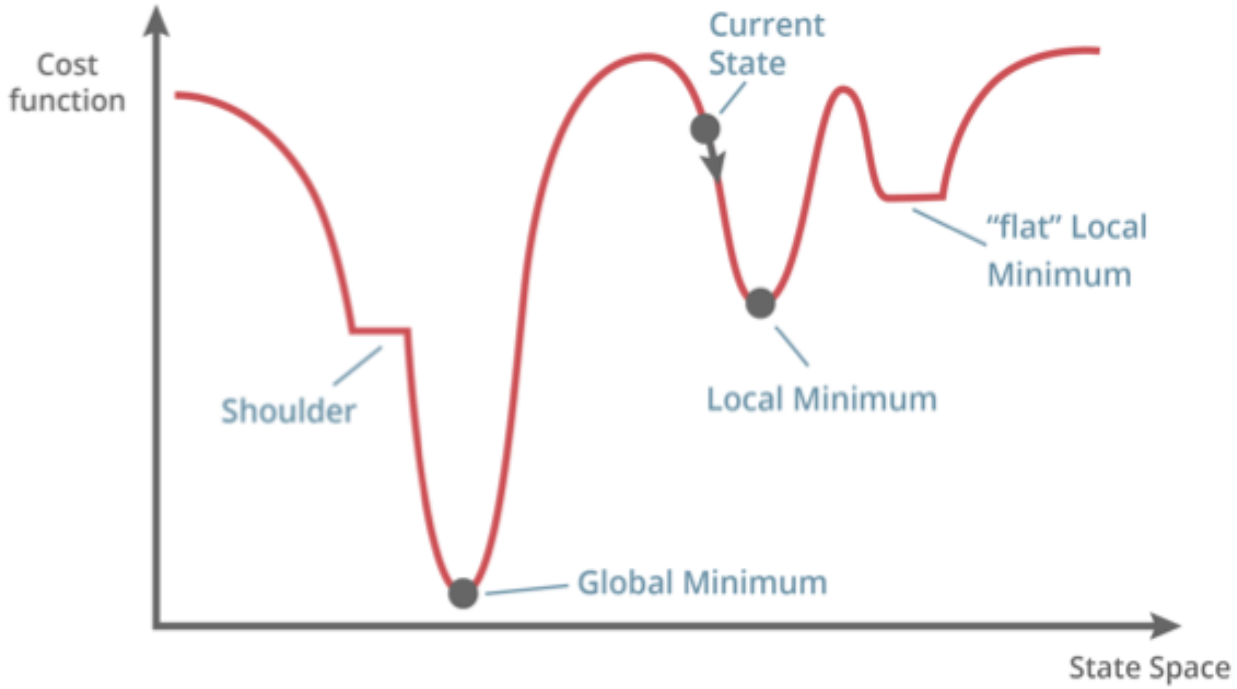
- Variable selection: randomly select any conflicted variable
- Value selection by min-conflicts heuristic
  - choose value that violates the fewest constraints

Flat regions and local optima



- Sometimes, have to go sideways or even backwards in order to make progress towards the actual solution.

Inverted View



- When we are minimising violated constraints, it makes sense to think of starting at the top of a ridge and climbing down into the valleys.

Simulated Annealing

随机的

- **Stochastic** hill climbing based on difference between evaluation of previous state (h_0) and new state (h_1).
  - If $h_1 < h_0$, definitely make the change
  - Otherwise, make the change with probability

$$e^{-(h1 - h0)/T}$$

where T is a "temperature" parameter.

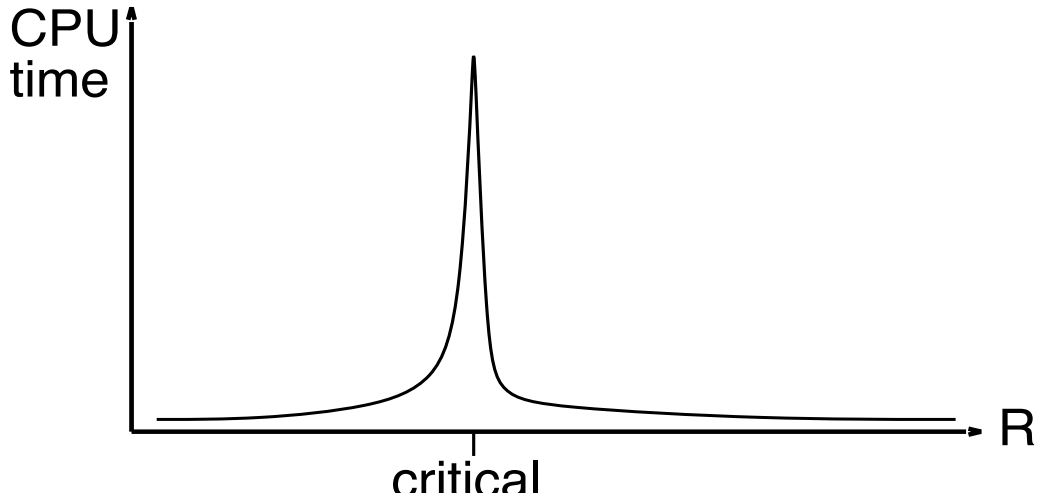
- Reduces to ordinary hill climbing when $T = 0$
- Becomes totally random search as $T \rightarrow \infty$
- Sometimes, we gradually decrease the value of T during the search

Phase transition in CSP's

Given random initial state, hill climbing by min-conflicts with random restarts can solve n-queens in almost constant time for arbitrary n with high probability (e.g., n = 10,000,000).

In general, randomly-generated CSP's tend to be easy if there are very few or very many constraints. They become extra hard in a narrow range of the ratio

$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$



Summary

- CSPs are a special kind of problem:
  - states defined by values of a fixed set of variables
  - goal test defined by constraints on variable values
- Backtracking = depth-first search with one variable assigned per node
- Variable ordering and value selection heuristics help significantly
- Forward checking prevents assignments that guarantee later failure
- Constraint propagation (e.g., arc consistency) does additional work to constrain values and detect inconsistencies
- Iterative min-conflicts is usually effective in practice
- Simulated Annealing can help to escape from local optima