

# CSC384 Updates

---

- Remaining A1 Help Sessions:
  - Tuesday, January 30 11:00 am, Location TBD.
  - Thursday, February 1, 4:00 pm, Location TBD.
  - Friday, February 2, 11:00 am, Location TBD.
- The A1 TA, Andrew Perrault, is also available via Piazza for questions concerning the assignment. If you have a question of a personal nature, please email Andrew at `t1perrau teach dot cs dot utoronto dot ca` or an instructor; place A1 and 384 in the subject header of your message.

# Constraint Satisfaction Problems (CSPs)

---

*aka Backtracking Search*

Chapter 6 (R&N, 3rd edition)

- 6.1: Formalism
- 6.2: Constraint Propagation
- 6.3: Backtracking Search for CSP
- 6.4 is about local search which is a very useful idea but we won't cover it in class.

In R&N 2nd edition the comparable chapter is Chapter 5.  
Section 5.3 corresponds to section 6.4 in the 3rd Ed.

# Constraint Satisfaction Problems

---

- As we've discussed search problems, we've designed problem-specific state representations for our search routines to use.
- We've also generally been concerned not only to arrive at goal states, but to determine paths for our agent(s) to follow.
- We might, however, care less about paths and more about final goal states or configurations.
- We might also prefer general state representations that can take advantage of specialized search algorithms.
- We call search problems that can be formalized in this specialized way **CSPs, or Constraint Satisfaction Problems**.
- CSPs are search problems with a uniform, simple state representation that allows design of more efficient algorithms.
- Techniques for solving CSPs have many practical applications in industry.

# Constraint Satisfaction Problems

---

## State Representation:

factored into variables that can take on values

## Algorithms:

general purpose, enforce constraints on factors

## Main Idea:

eliminate chunks of search space by identifying value assignments for variables that **violate constraints**

# State Representation

---

**State Representation:** states are vectors of feature values

We have:

- A set of k **features** (or **variables**)
- Each variable has a **domain** of different values.
- A **state** is specified by an assignment of a value for each variable.
  - height = {short, average, tall},
  - weight = {light, average, heavy}
- A **partial state** is specified by an assignment of a value to some of the variables.

In CSPs, the problem is to search for a set of values for the features (variables) so that the values satisfy some conditions (constraints).

- i.e., a goal state specified as conditions on the vector of feature values.

# Example: Sudoku

---

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

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

# Example: Sudoku

---

- 81 **features** or **variables**, each representing the value of a cell.
- **Domain of values**: a fixed set of possible values cells. For cells that are already filled in, the domain is the singleton set containing the predefined value. The domain of other variables is the set  $\{1-9\}$
- **State**: any completed board given by specifying the value in each cell (1-9, or blank).
- **Partial state**: some incomplete filling out of the board.
- **Solution**: a value for each cell satisfying the constraints:
  - no cell in the same column can have the same value.
  - no cell in the same row can have the same value.
  - no cell in the same sub - square can have the same value.

# Example: 8-Puzzle

---

- **Variables:** 9 variables  $\text{Cell}_{1,1}, \text{Cell}_{1,2}, \dots, \text{Cell}_{3,3}$
- **Values:**  $\{\text{'B'}, 1, 2, \dots, 8\}$
- **State:** An assignment of domain values to each “ $\text{Cell}_{i,j}$ ”

*This is only one of many ways to specify the state.*

2	3	7
6	4	8
5	1	



# Constraint Satisfaction Problems

---

- Notice that in these problems some settings of the variables are **illegal**.
  - In Sudoku, we can't have the same number in any column, row, or sub-square.
  - In the 8 puzzle each variable must have a distinct value (same tile can't be in two places)

# Constraint Satisfaction Problems

---

In many practical problems finding which setting of the feature variables yields a legal state is difficult.

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

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

We want to find a state (setting of the variables) that satisfies certain constraints.

# Constraint Satisfaction Problems

---

In Suduko: The variables that form

- a column must be distinct
- a row must be distinct
- a sub-square must be distinct.

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

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

# Constraint Satisfaction Problems

---

- Note that in these problems we **do not care** about the sequence of moves needed to get to a goal state.
- We only care about finding a feature vector (a setting of the variables) that satisfies the goal.
  - i.e., a setting of the variables that satisfies some constraints.
- In contrast, for the 8-puzzle, we care about the sequence of moves needed to move the tiles into a goal state configuration.

# Example: Scheduling

---

We want to schedule a time and a space for each final exam so that

- No student is scheduled to take more than one final at the same time.
- The space allocated has to be available at the time set.
- The space has to be large enough to accommodate all of the students taking the exam.

# Example: Scheduling

---

## Scheduling Problem **Variables:**

We use the index “i” to uniquely identify exams -the “i-th exam”.

## Scheduling **TIME:**

- $T_1, \dots, T_m$ :  $T_i$  is a variable representing the scheduled time for the i-th final exam.
- Assume domains are fixed to  $\{\text{FriAm}, \text{TuesPm}, \text{MonAm}, \text{MonPm}, \dots, \text{FriPm}\}$ .

## Scheduling **SPACE :**

- $S_1, \dots, S_m$ :  $S_i$  is the space variable for the i-th final.
- Domain of  $S_i$  are all rooms big enough to hold the i-th final.

# Example: Scheduling

---

Want to find an assignment of values to each variable (times, rooms for each final), subject to the **constraints**:

1. For all pairs of finals  $i, j$  ( $i \neq j$ ) such that there is a student taking both:

$$T_i \neq T_j$$

2. For all pairs of finals  $i, j$  ( $i \neq j$ ):

$$T_i \neq T_j \text{ or } S_i \neq S_j$$

*either final  $i$  and  $j$  are not scheduled at the same time, or if they are not in the same space.*

# Formalization of a CSP

---

A CSP consists of

- A set of **variables**  $V_1, \dots, V_n$
- A(finite) **domain** of **possible values**  $\text{Dom}[V_i]$  for each variable.
- A set of **constraints**  $C_1, \dots, C_m$ .
- A **solution** to a CSP is an assignment of a value to all of the variables such that ***every constraint is satisfied***.
- A CSP is *unsatisfiable* if no solution exists.



# Formalization of a CSP

---

- Each variable can be assigned any value from its domain  
 $V_i = d$  where  $d \in \text{Dom}[V_i]$
- Each constraint  $C$ 
  - Has a set of variables it operates over, called its **scope**  
E.g.,  $C(V_1, V_2, V_4)$  is a constraint over the variables  $V_1, V_2$ , and  $V_4$ . Its **scope** is  $\{V_1, V_2, V_4\}$
  - Given an assignment to variables the constraint returns:
    - True** if the assignment satisfies the constraint
    - False** if the assignment falsifies the constraint.

# Types of Constraints

---

- **Unary** Constraints (over one variable)
  - e.g.  $C(X): X=2$ ;  $C(Y): Y>5$
- **Binary** Constraints (over two variables)
  - e.g.  $C(X,Y): X+Y<6$
- **Higher-order** constraints: over 3 or more variables.

# Solutions

---

A solution is an assignment of a value to each of the variables such that every constraint is satisfied

# Formalization of a CSP

---

We can specify the constraint with a table

V1	V2	V4	C(V1,V2,V4)
1	1	1	False
1	1	2	False
1	2	1	False
1	2	2	False
2	1	1	True
2	1	2	False
2	2	1	False
2	2	2	False
3	1	1	False
3	1	2	True
3	2	1	True
3	2	2	False

$C(V1, V2, V4)$  with  $\text{Dom}[V1] = \{1,2,3\}$  and  $\text{Dom}[V2] = \text{Dom}[V4] = \{1, 2\}$

# Formalization of a CSP

---

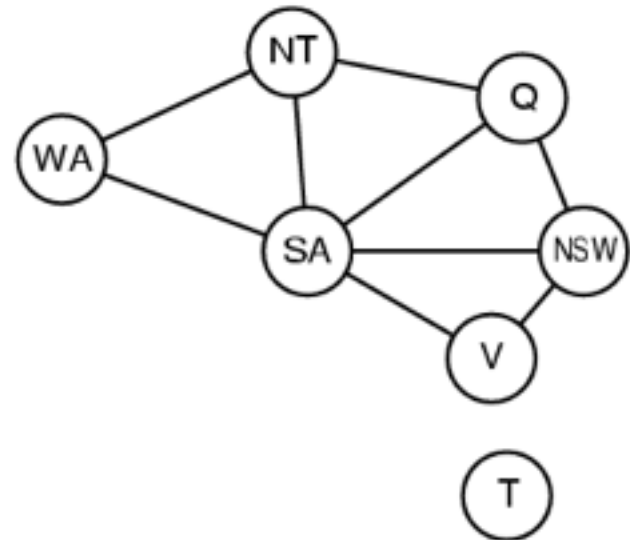
Or, often, more compactly, as an expression

V1	V2	V4	C(V1,V2,V4)
1	1	1	False
1	1	2	False
1	2	1	False
1	2	2	False
2	1	1	True
2	1	2	False
2	2	1	False
2	2	2	False
3	1	1	False
3	1	2	True
3	2	1	True
3	2	2	False

$$C(V1, V2, V4) : V1 = V2 + V4$$

# Representing CSP as a graph

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

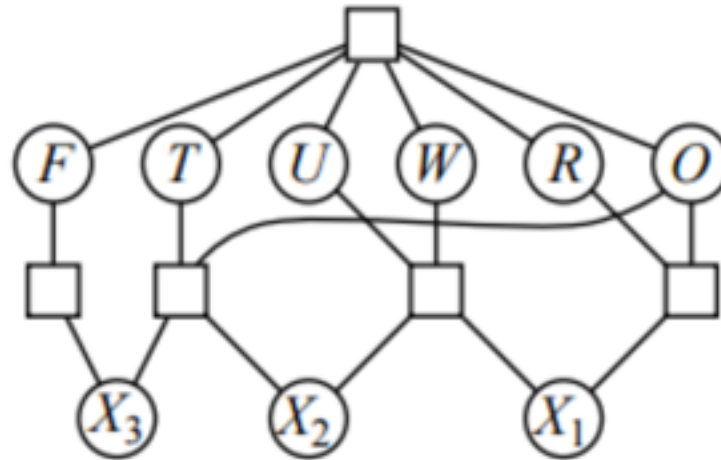


- **Variables:** WA, NT, Q, NSW, V, SA, T
- **Domains:**  $D_i = \{\text{red, green, blue}\}$
- **Constraints:** adjacent regions must have different colours  
e.g.,  $WA \neq NT$ , or  $(WA, NT)$  in  $\{(\text{red, green}), (\text{red, blue}), (\text{green, red}), (\text{green, blue}), (\text{blue, red}), (\text{blue, green})\}$

# Representing CSP as a graph

- Higher Order Constraints

$$\begin{array}{r} T \ W \ O \\ + \ T \ W \ O \\ \hline F \ O \ U \ R \end{array}$$



**Variables** F,T,U,W,R,O,X1,X2,X3

**Domains**  $D_i = [0-9]$

**Constraints:**

$$O + O = R + 10 * X_1$$

$AllDiff(F, T, U, W, R, O)$

# Example: Sudoku

---

- **Variables:**  $V_{11}, V_{12}, \dots, V_{21}, V_{22}, \dots, V_{91}, \dots, V_{99}$
- **Domain** specification:
  - $\text{Dom}[V_{ij}] = \{1-9\}$  for empty cells
  - $\text{Dom}[V_{ij}] = \{k\}$  a fixed value  $k$  for filled cells.
- **Constraints:**
  - Row constraints:
    - $\text{All-Diff}(V_{11}, V_{12}, V_{13}, \dots, V_{19})$
    - $\text{All-Diff}(V_{21}, V_{22}, V_{23}, \dots, V_{29})$
    - $\dots, \text{All-Diff}(V_{91}, V_{92}, \dots, V_{99})$
  - Column Constraints:
    - $\text{All-Diff}(V_{11}, V_{21}, V_{31}, \dots, V_{91})$
    - $\text{All-Diff}(V_{12}, V_{22}, V_{32}, \dots, V_{92})$
    - $\dots, \text{All-Diff}(V_{19}, V_{29}, \dots, V_{99})$
  - Sub-Square Constraints:
    - $\text{All-Diff}(V_{11}, V_{12}, V_{13}, V_{21}, V_{22}, V_{23}, V_{31}, V_{32}, V_{33})$



# Example: Sudoku

---

- Each of these constraints is over 9 variables, and they are all the same constraint:
  - Any assignment to these 9 variables such that each variable has a different value satisfies the constraint.
  - Any assignment where two or more variables have the same value falsifies the constraint.
- This is a special kind of constraint called an **ALL-Diff** constraint.
  - ALL-Diff( $V_1, \dots, V_n$ ) could also be encoded as a set of binary not-equal constraints between all possible pairs of variables:  
 $V_1 \neq V_2, V_1 \neq V_3, \dots, V_2 \neq V_1, \dots, V_n \neq V_1, \dots, V_n \neq V_{n-1}$
  - ALL-Diff constraints are common in many CSP applications and special purpose solvers have been developed for evaluating ALL-Diff constraints and also for exploiting ALL-Diff constraints in the context of constraint propagation, which we'll see later on.

# Example: Sudoku

---

- Thus Sudoku has 3x9 ALL-Diff constraints:
  - one over each set of variables in the same row
  - one over each set of variables in the same column
  - and one over each set of variables in the same sub-square.

# Solving CSPs

---

- Because CSPs do not require finding a paths (to a goal), it is best solved by a specialized version of depth-first search.
- Key intuitions:
  - We can build up to a solution by searching through the space of partial assignments.
  - Order in which we assign variables does not matter - eventually they all have to be assigned. We can decide on a suitable value for one variable at a time!

This is the key idea of backtracking search.

- If we falsify a constraint during the process of building up a solution, we can immediately reject the current partial assignment:
  - All extensions of this partial assignment will falsify that constraint, and thus none can be solutions.

# CSP as a Search Problem

---

A CSP could be viewed as a more traditional search problem

- **Initial state:** empty assignment
- **Successor function:** a value is assigned to any unassigned variable, which does not cause any constraint to return false.
- **Goal test:** the assignment is complete

# Backtracking (BT) Algorithm


---

- In the slides that follows, we present a generic backtracking algorithm:
  - We pick a variable, assign it a value, test the constraints that we can. **If a constraint is unsatisfied we backtrack,** otherwise we set another variable. When all the variables are set, we're done.
- Later in this unit, we refine this algorithm to include some constraint propagation (inference). More on this later!
- there are clever ways to pick what variable to assign and also what value to assign to it! Can you think of any? Think about what you do when you play Sudoku!

# Backtracking (BT) Algorithm

---

BT(Level)

  If all variables assigned 

    PRINT Value of each Variable

    RETURN or EXIT (RETURN for more solutions)  
                    (EXIT for only one solution)

  V := **PickUnassignedVariable()**

  Assigned[V] := TRUE

  for d := each member of Domain(V) (the domain values of V)

    Value[V] := d

    ConstraintsOK = TRUE

    for each constraint C such that

      a) V is a variable of C and

      b) **all other variables of C are assigned:**

*;may occur rarely high in tree*

    IF C is **not** satisfied by the set of current  
    assignments:

      ConstraintsOK = FALSE

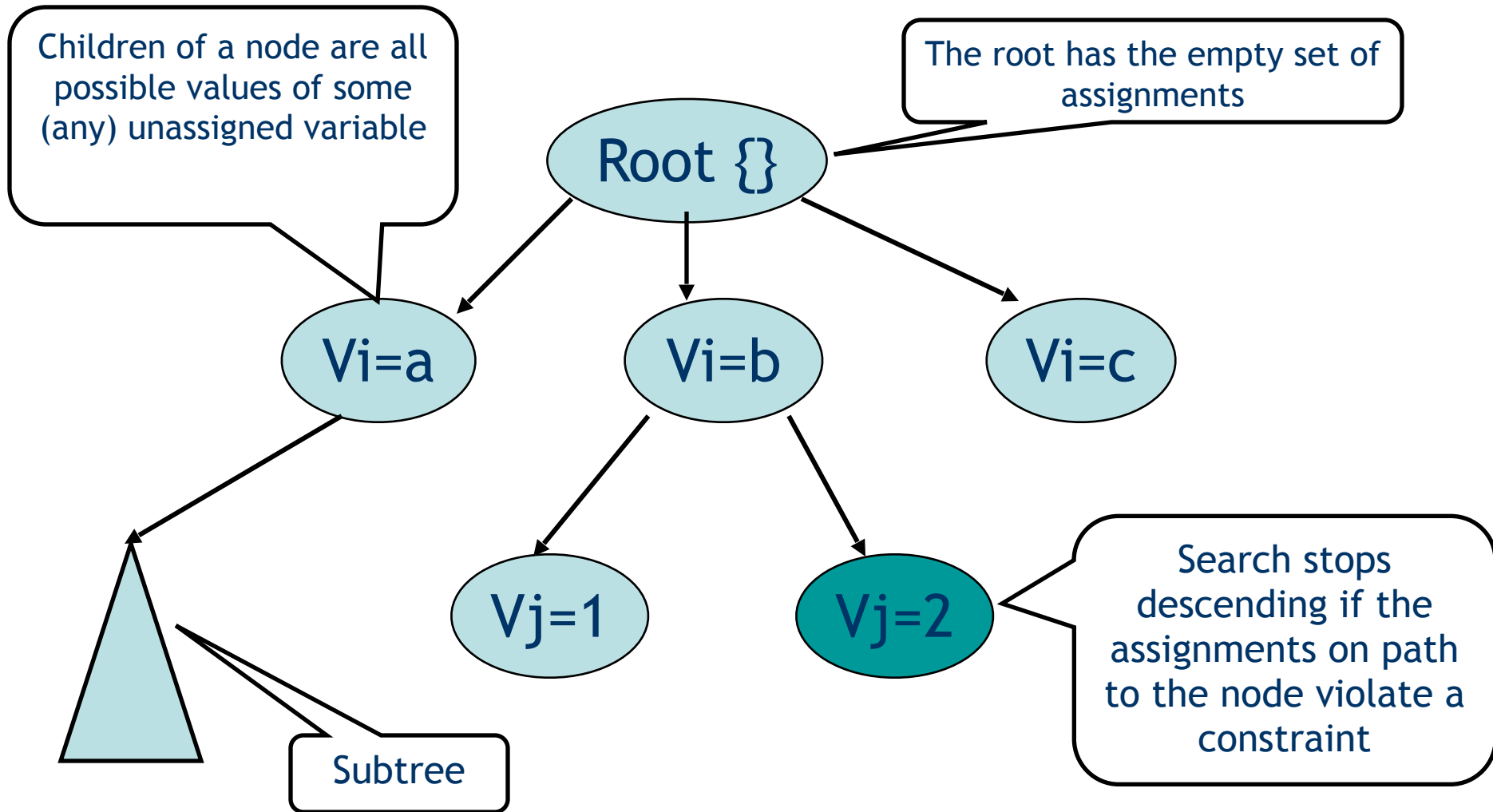
  If ConstraintsOk == TRUE:

    BT(Level+1)

  Assigned[V] := FALSE //UNDO as we have tried all of V's values  
  return

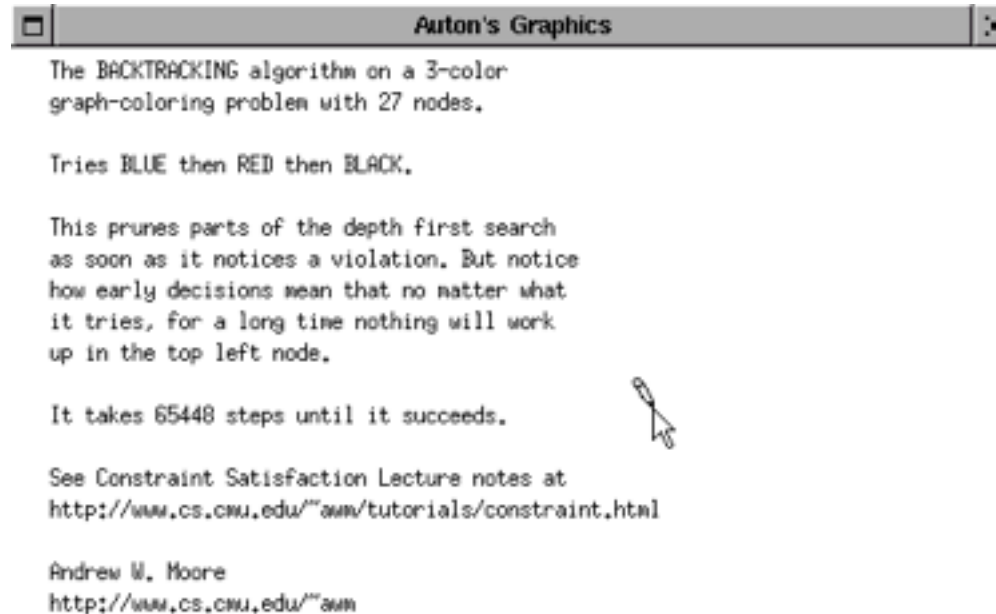
# Backtracking (BT) Search

The algorithm searches a tree of partial assignments.



# Backtracking (BT) example

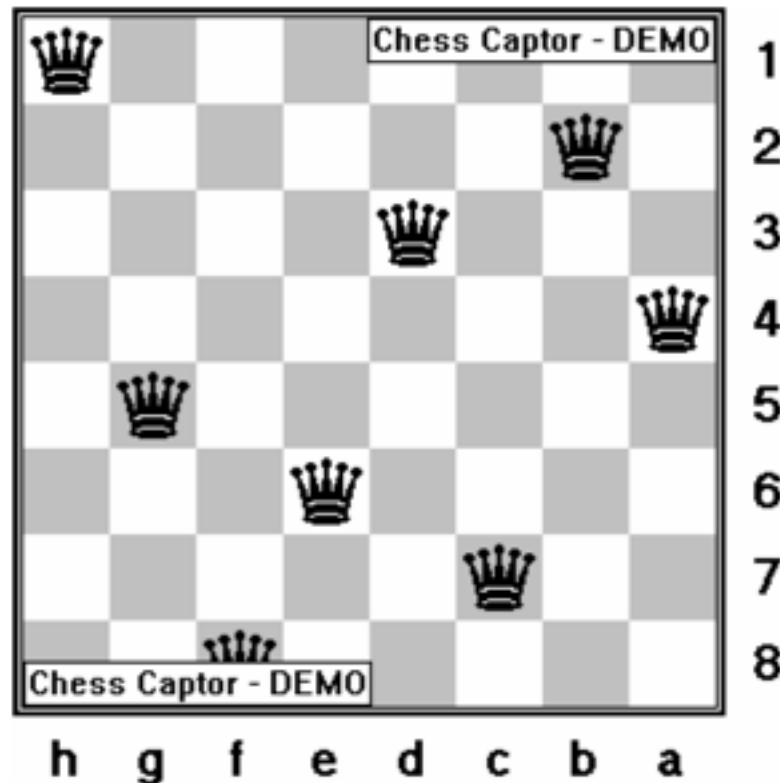
---





# Example: N-Queens

- Place N Queens on an N X N chess board so that no Queen can attack any other Queen.



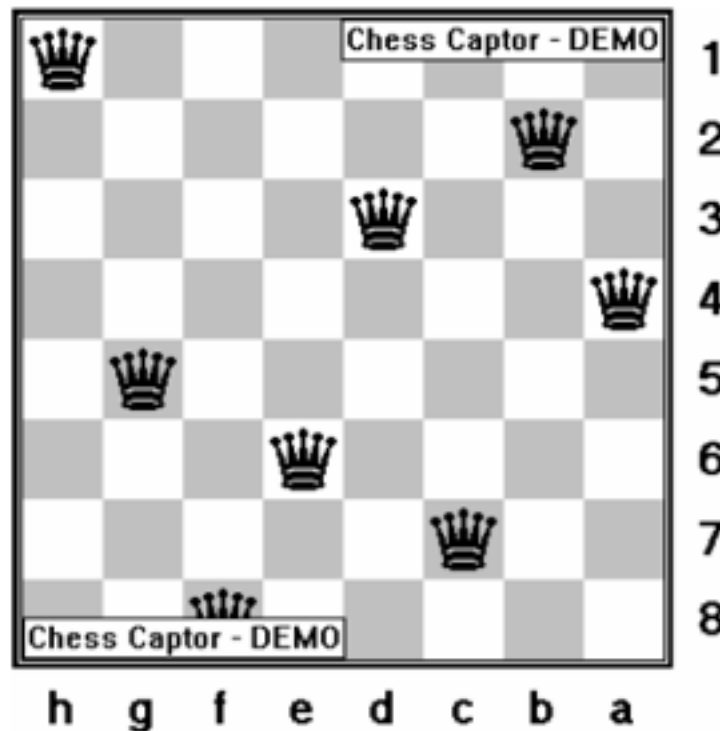
# Example: N-Queens

---

- Problem formulation:
  - N variables (N queens)
  - $N^2$  values for each variable representing the positions on the chessboard
    - Value  $i$  is  $i$ 'th cell counting from the top left as 1, going left to right, top to bottom.

# Example: N-Queens

$Q1 = 1, Q2 = 15, Q3 = 21, Q4 = 32, Q5 = 34,$   
 $Q6 = 44, Q7 = 54, Q8 = 59$



# Example: N-Queens

---

- This representation has  $(N^2)^N$  states (different possible assignments in the search space)
  - For 8-Queens:  $64^8 = 281,474,976,710,656$
- Is there a better way to represent the N-queens problem?
  - We know we cannot place two queens in a single row
  - We can exploit this fact in the choice of the CSP representation

# Example: N-Queens

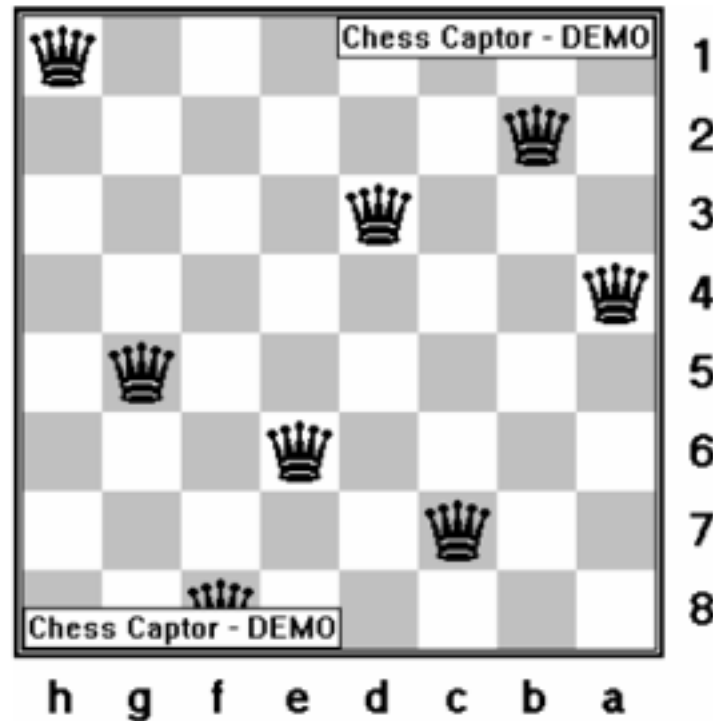
---

- Better Model:
  - N variables  $Q_i$ , one per row.
  - Value of  $Q_i$  is the column the Queen in row  $i$  is placed; possible values  $\{1, \dots, N\}$ .
- This representation has  $N^N$  states:
  - For 8-Queens:  $8^8 = 16,777,216$
- **The choice of a representation can make the problem solvable or unsolvable!**

# Example: N-Queens

---

$Q1 = 1, Q2 = 7, Q3 = 5, Q4 = 8, Q5 = 2, Q6 = 4, Q7 = 6, Q8 = 3$



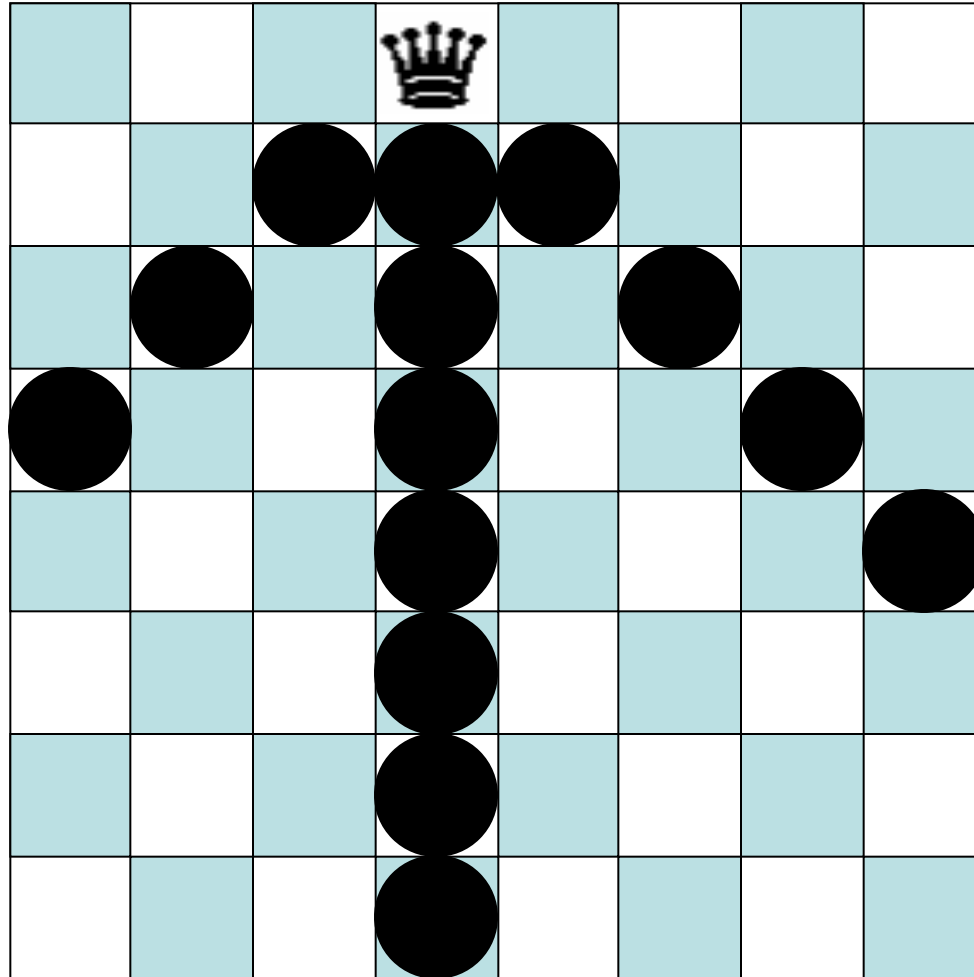
# Example: N-Queens

---

- Constraints:
  - Can't put two Queens in same column  
 $Q_i \neq Q_j$  for all  $i \neq j$
  - Diagonal constraints  
 $\text{abs}(Q_i - Q_j) \neq \text{abs}(i - j)$ 
    - i.e., the difference in the values assigned to  $Q_i$  and  $Q_j$  can't be equal to the difference between  $i$  and  $j$ .

# Example: N-Queens

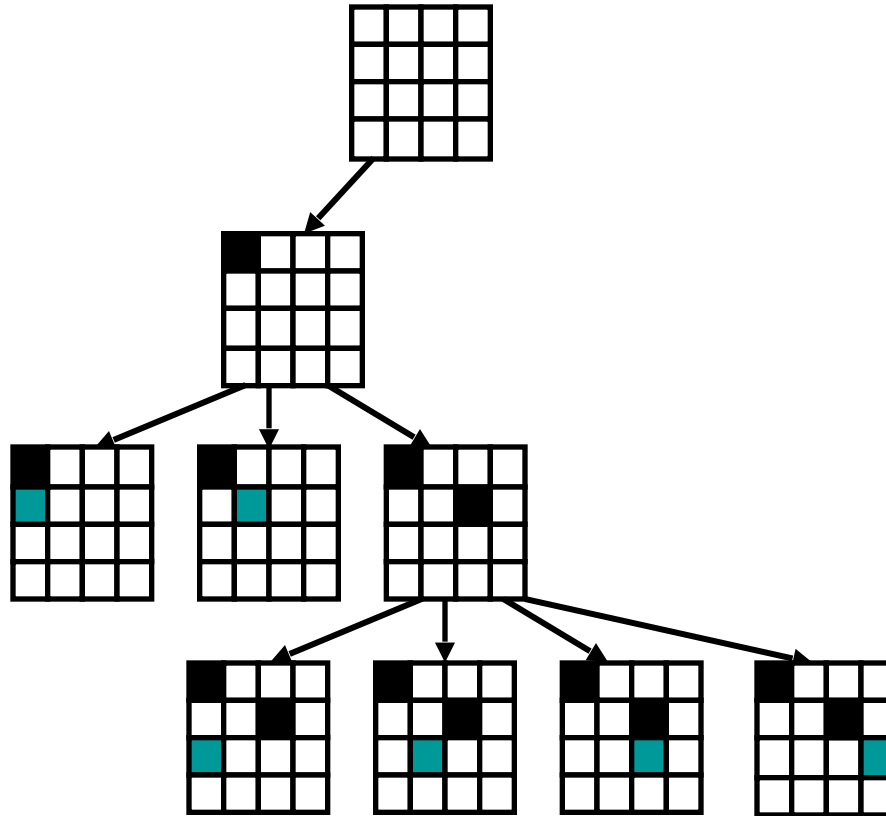
---



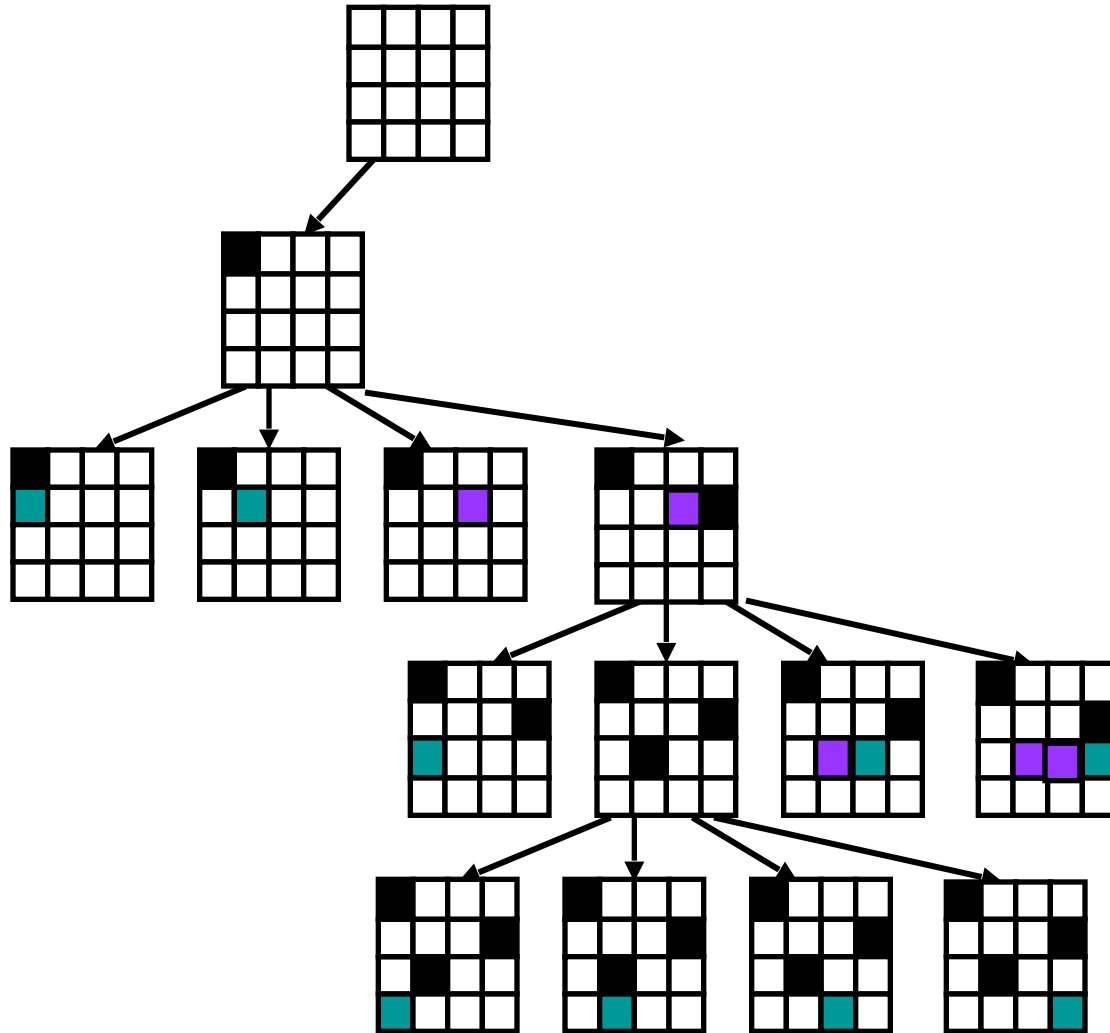


# Example: N-Queens

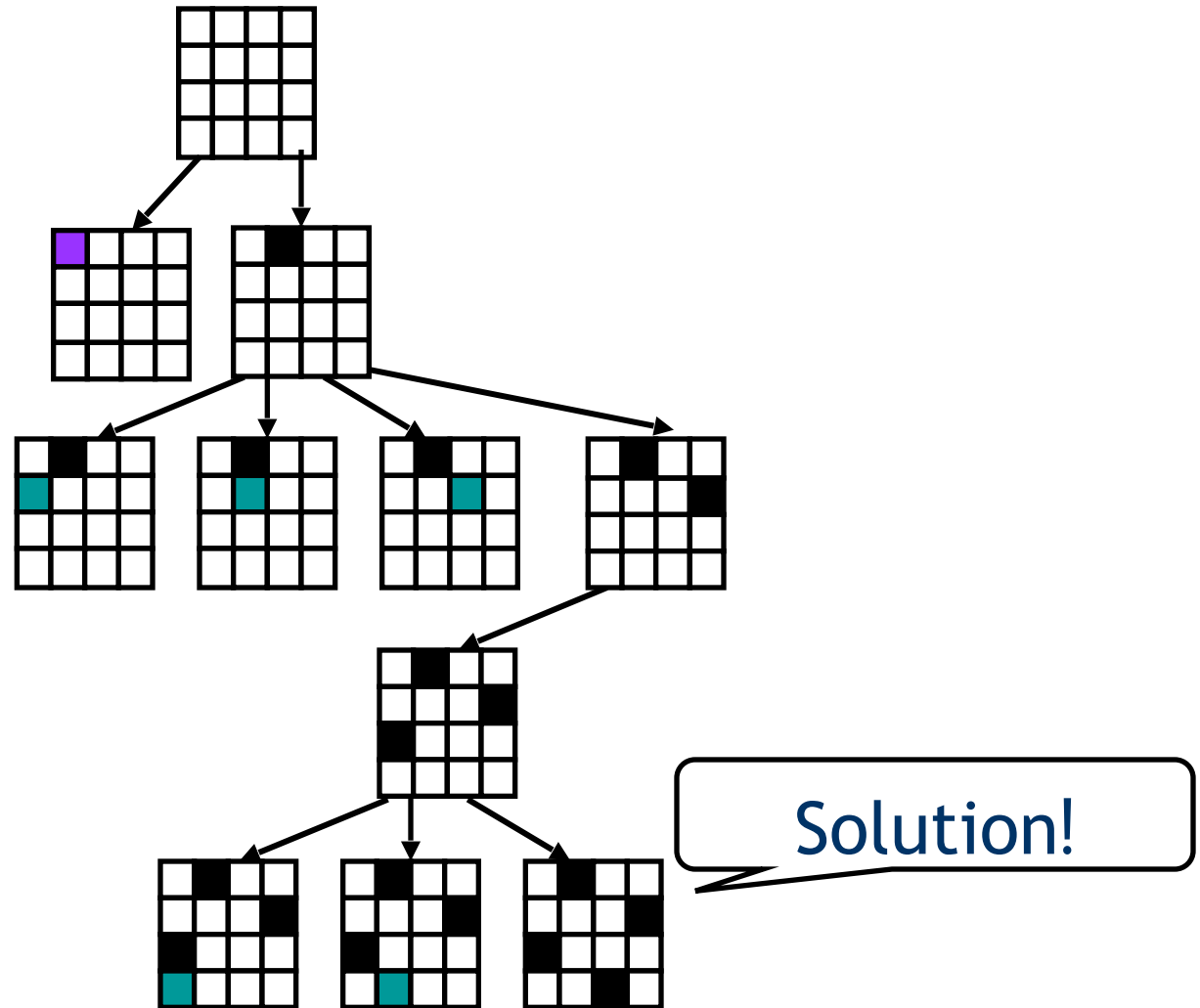
---



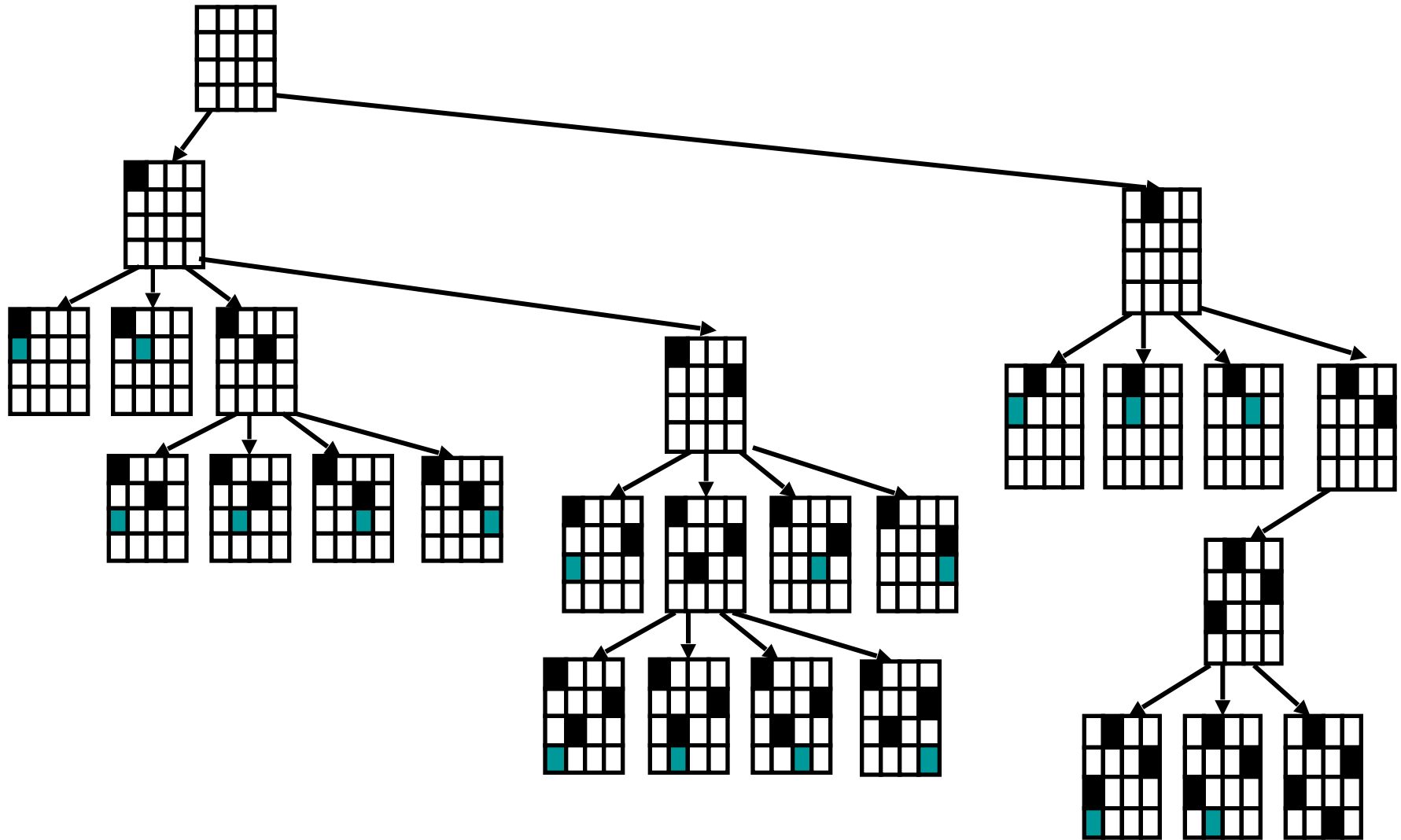
# Example: N-Queens



# Example: N-Queens



# Example: N-Queens Backtracking Search



# Problems with Plain Backtracking

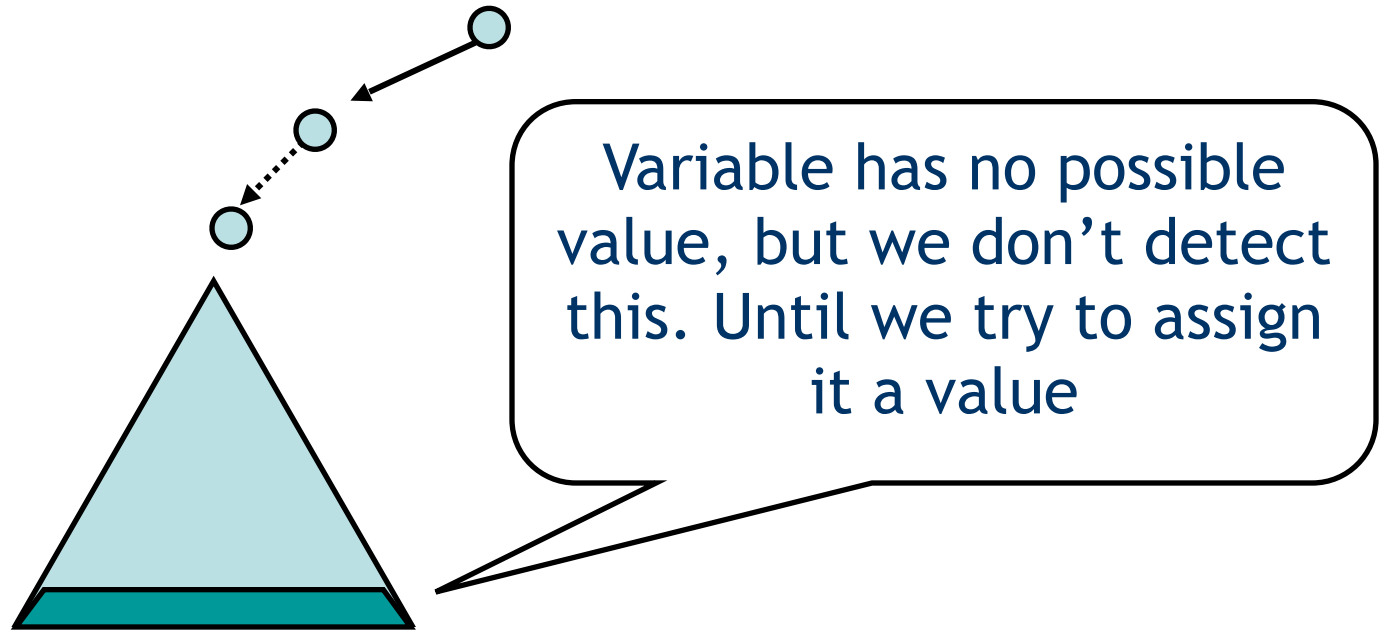
---

1	2	3						
						4	5	6
		7						
		8						
		9						

Sudoku: The 3,3 cell has no possible value.

# Problems with Plain Backtracking

In the backtracking search we won't detect that the (3,3) cell has no possible value until **all** variables of the row/column (involving row or column 3) or the sub-square constraint (first sub-square) **are assigned**. So we have the following situation:



Some tools to ameliorate this problem:

**constraint propagation, variable and value ordering heuristics**

# Constraint Propagation

---

- Constraint propagation refers to the technique of “looking ahead” at the yet unassigned variables in the search .
- Try to detect obvious failures: “obvious” means things we can test/detect efficiently.
- Even if we don’t detect an obvious failure we might be able to eliminate some possible part of the future search.

# Constraint Propagation

---

- Propagation has to be applied during the search; potentially at every node of the search tree.
- Propagation itself is an inference step that needs some resources (in particular time)
  - If propagation is slow, this can slow the search down to the point where using propagation makes finding a solution take longer!
  - There is always a tradeoff between searching fewer nodes in the search, and having a higher nodes/second processing rate.
- We will look at two main types of propagation:  
**Forward Checking & Generalized Arc Consistency**



# Constraint Propagation: Forward Checking

---

- Forward checking is an extension of backtracking search that employs a “modest” amount of propagation (look ahead).
- When a variable is instantiated we check all constraints that have **only one un-instantiated variable** remaining.
- For that un-instantiated variable, we check all of its values, pruning those values that violate the constraint.

# Forward Checking Algorithm

---

For a single constraint C:

`FCCheck(C, x)`

*// C is a constraint with all its variables already  
// assigned, except for variable x.*

for d := each member of CurDom[x]

IF making x = d together with previous assignments  
to variables in scope C **falsifies** C

THEN **remove d** from CurDom[x]

IF CurDom[x] = {} then return **DWO** (**D**omain **W**ipe **O**ut)

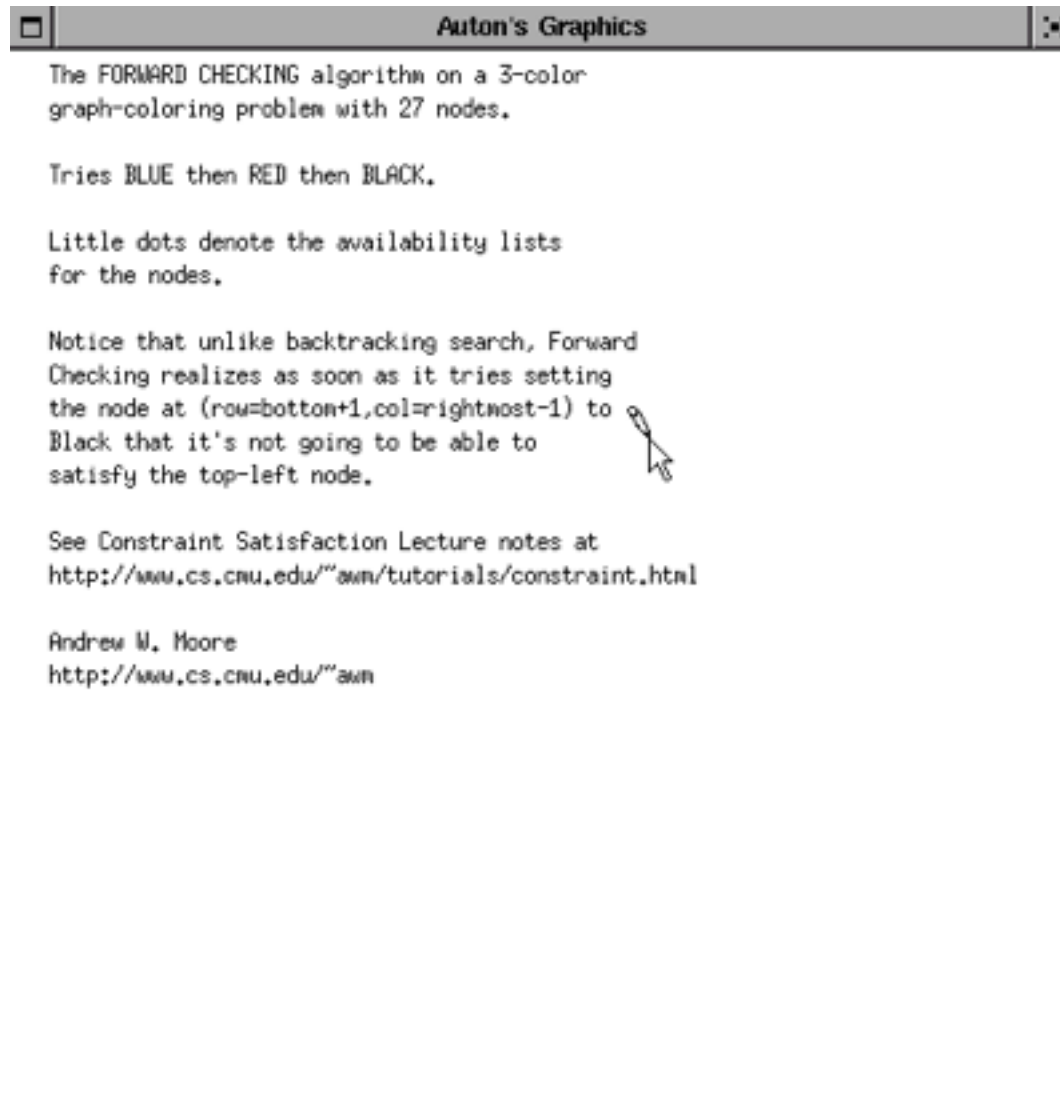
ELSE return ok

# Forward Checking Algorithm

---








```
FC(Level) /*Forward Checking Algorithm */
    If all variables are assigned
        PRINT Value of each Variable
        RETURN or EXIT (RETURN for more solutions)
                        (EXIT for only one solution)
    V := PickAnUnassignedVariable()
    Assigned[V] := TRUE
    for d := each member of CurDom(V)
        Value[V] := d
        DWOccured:= False
        for each constraint C over V such that
            a) C has only one unassigned variable X in its scope
            if(FCCheck(C,X) == DWO) /* X domain becomes empty*/
                DWOoccurred:= True
                break /* stop checking constraints */
        if(not DWOoccured) /*all constraints were ok*/
            FC(Level+1)
        RestoreAllValuesPrunedByFCCheck()
    Assigned[V] := FALSE //undo since we have tried all of V's values
    return;
```

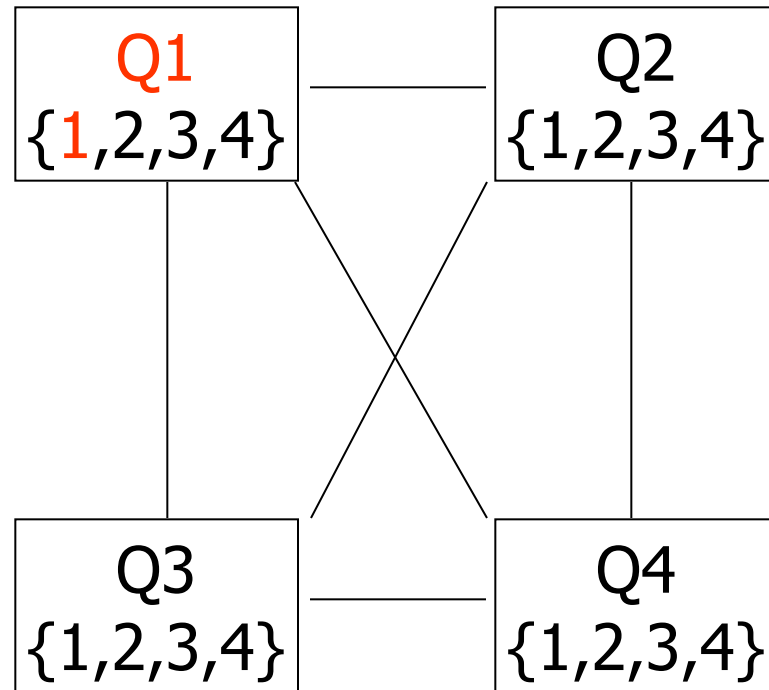
# Forward Checking Example



# 4-Queens Problem

- Encoding with  $Q1, \dots, Q4$  denoting a queen per row
  - cannot put two queens in same column

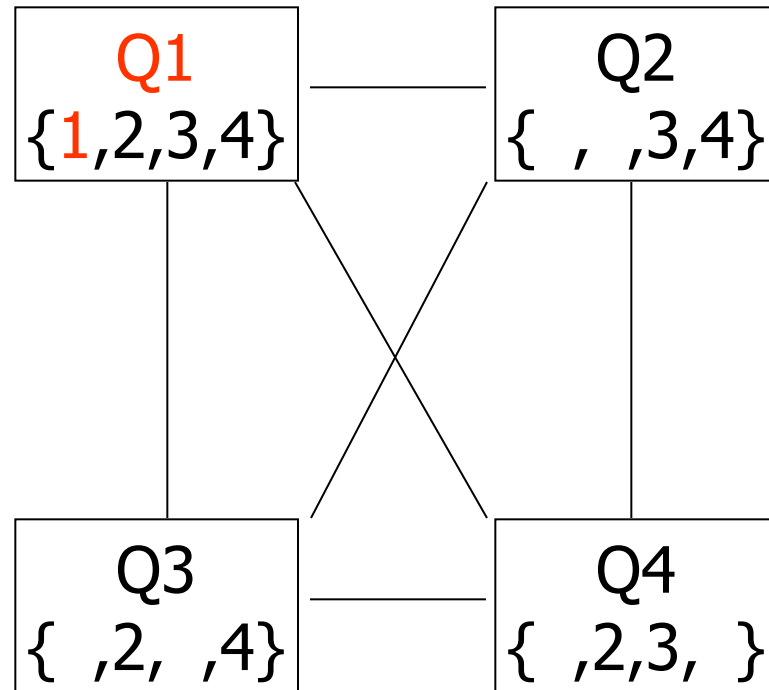
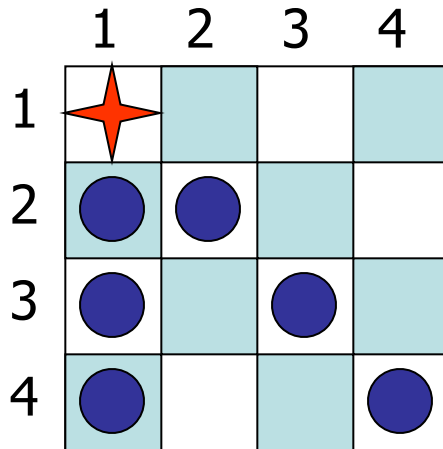
	1	2	3	4
1				
2				
3				
4				



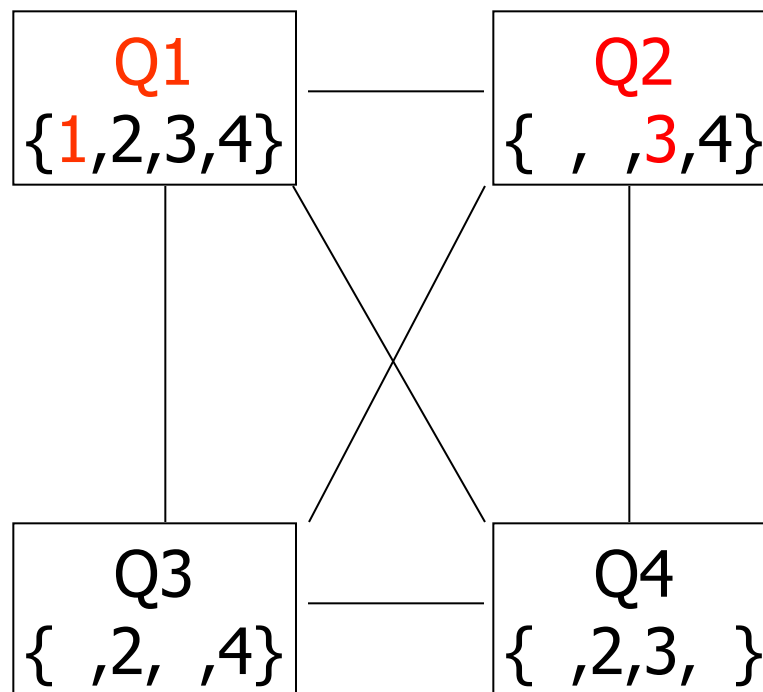
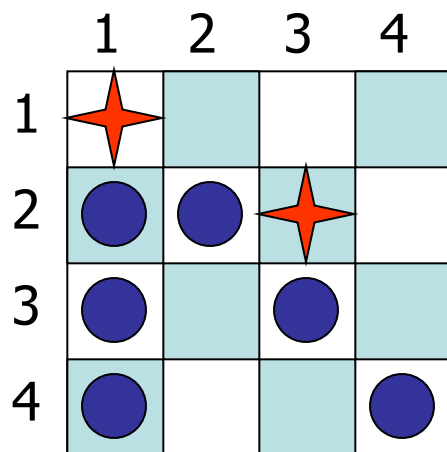
# 4-Queens Problem

Forward checking reduced the domains of all variables that are involved in a constraint with one un-instantiated variable:

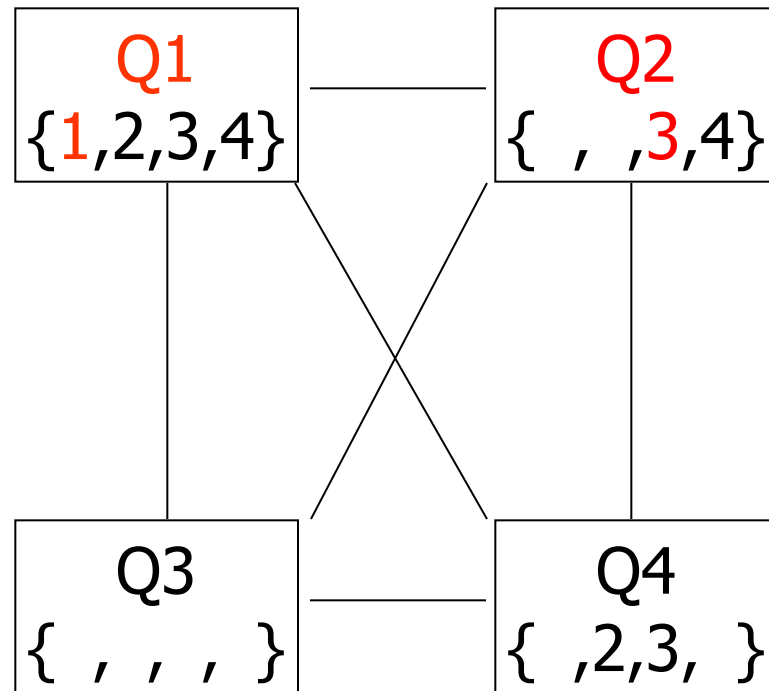
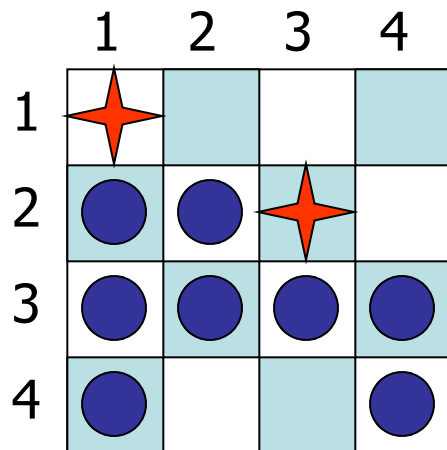
- Here all of Q2, Q3, Q4



# 4-Queens Problem



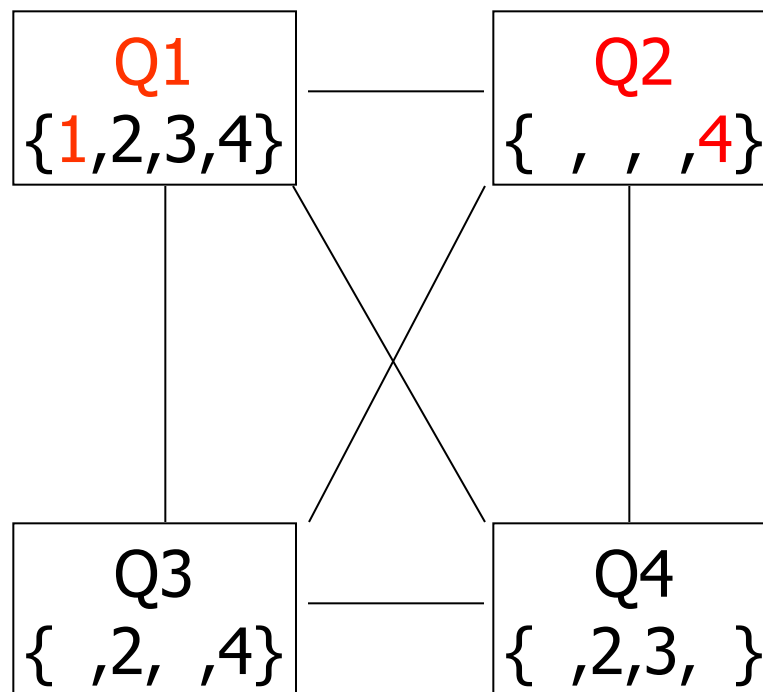
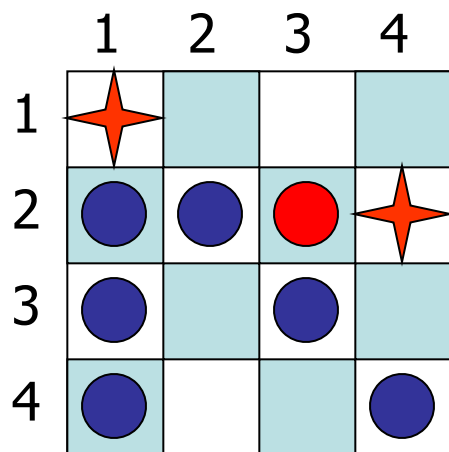
# 4-Queens Problem














DWO

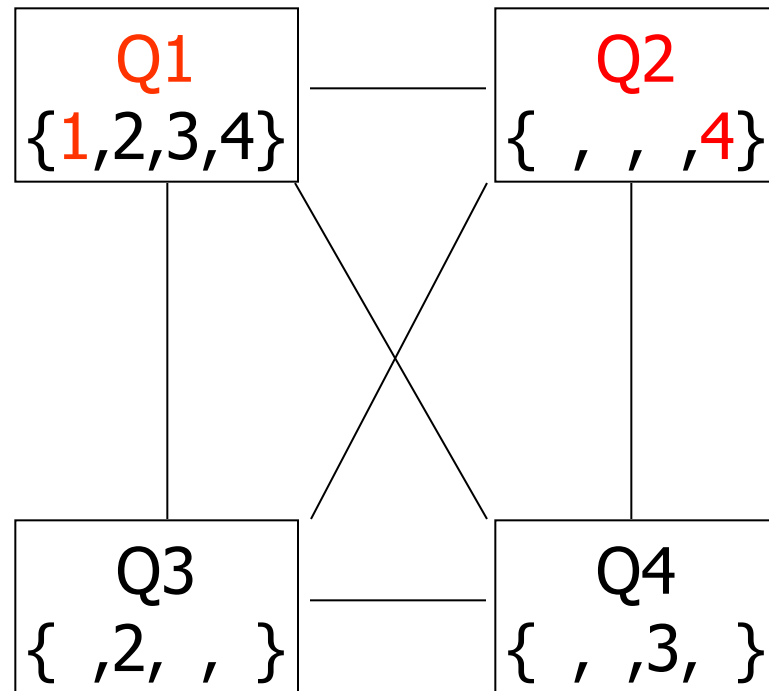


# 4-Queens Problem

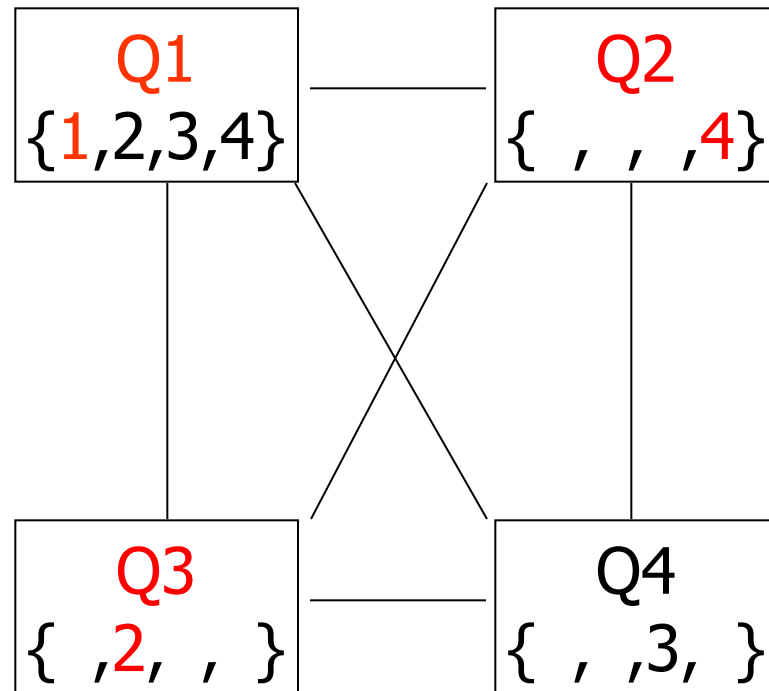
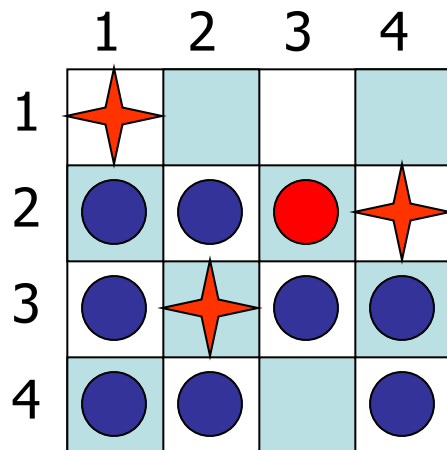


# 4-Queens Problem

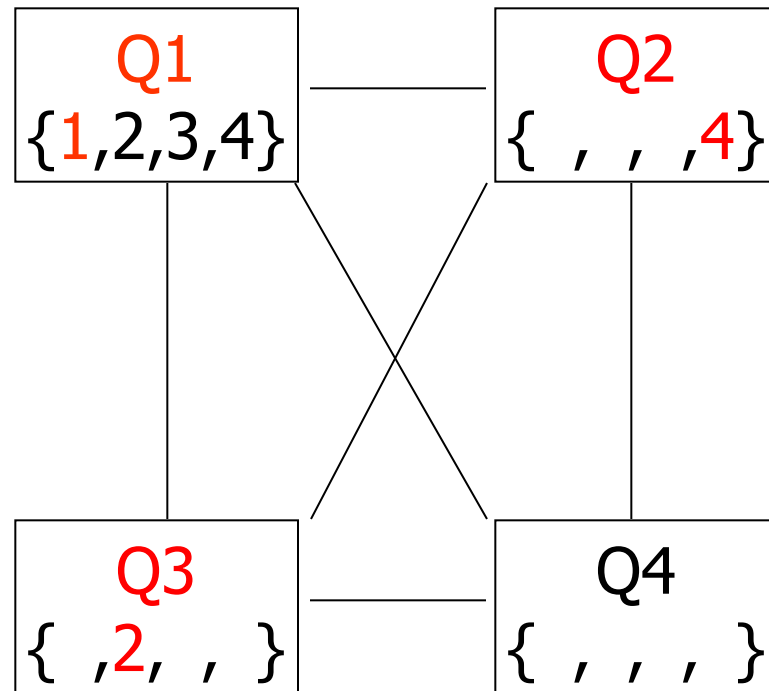
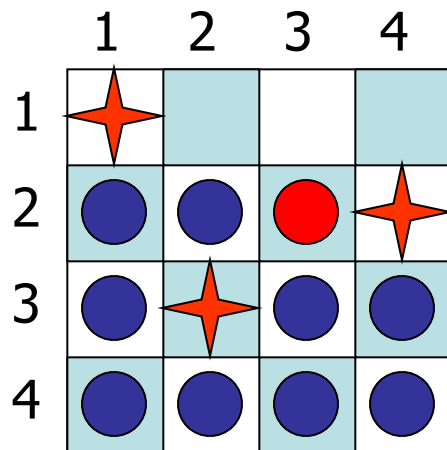
	1	2	3	4
1				
2				
3				
4				



# 4-Queens Problem



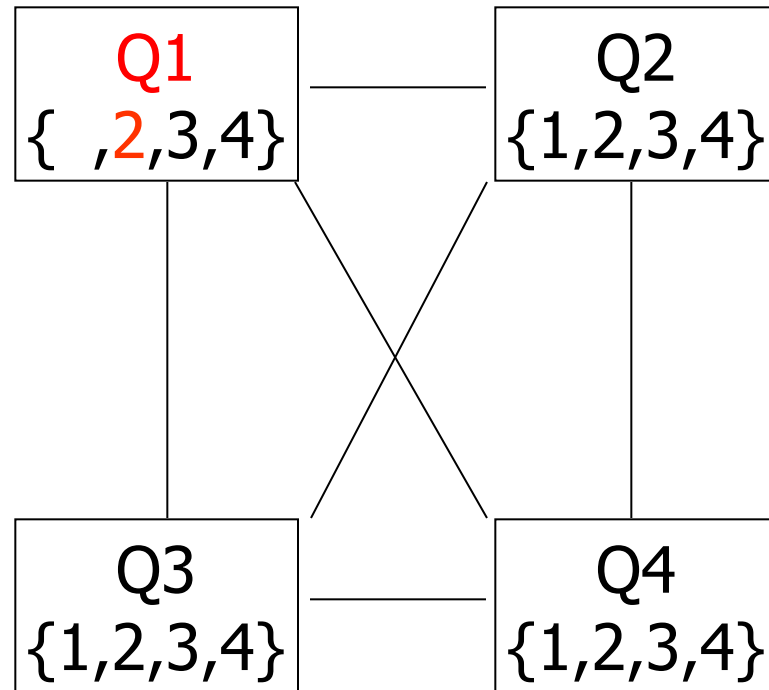
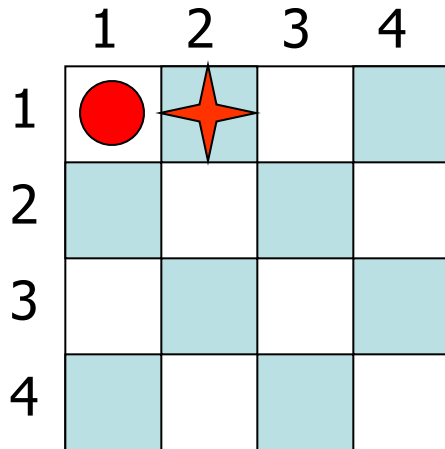
# 4-Queens Problem











DWO

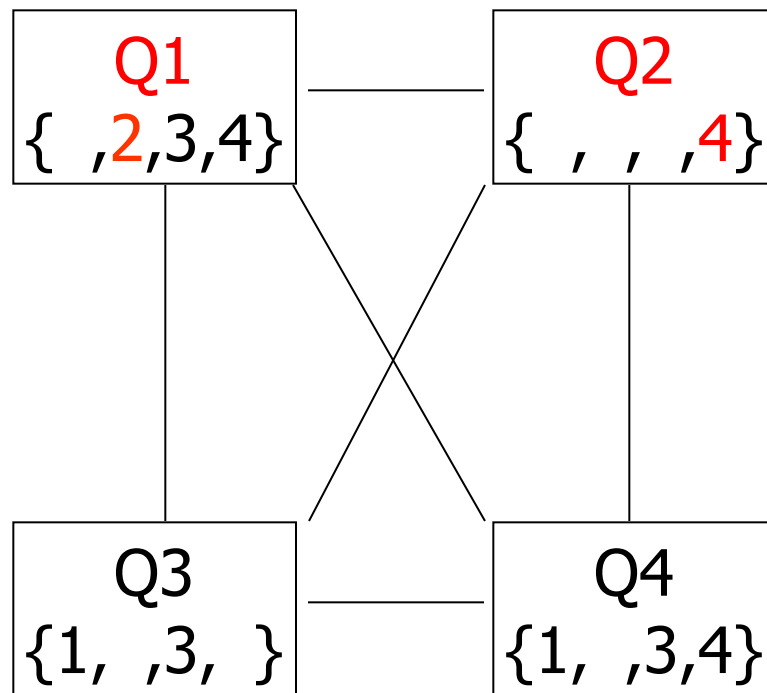
# 4-Queens Problem

Exhausted the subtree with  $Q1=1$ ; try now  $Q1=2$












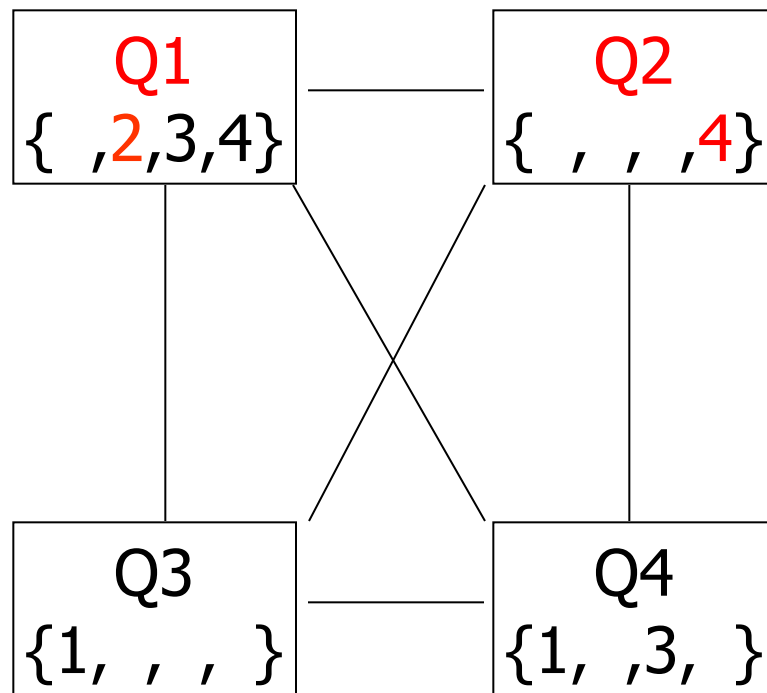
# 4-Queens Problem

	1	2	3	4
1				
2				
3				
4				














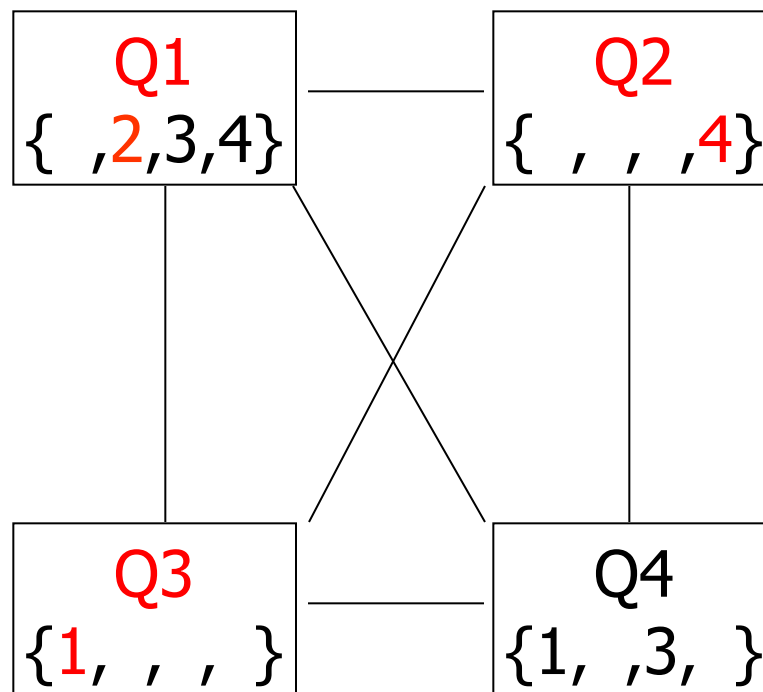
# 4-Queens Problem

	1	2	3	4
1				
2				
3				
4				















# 4-Queens Problem

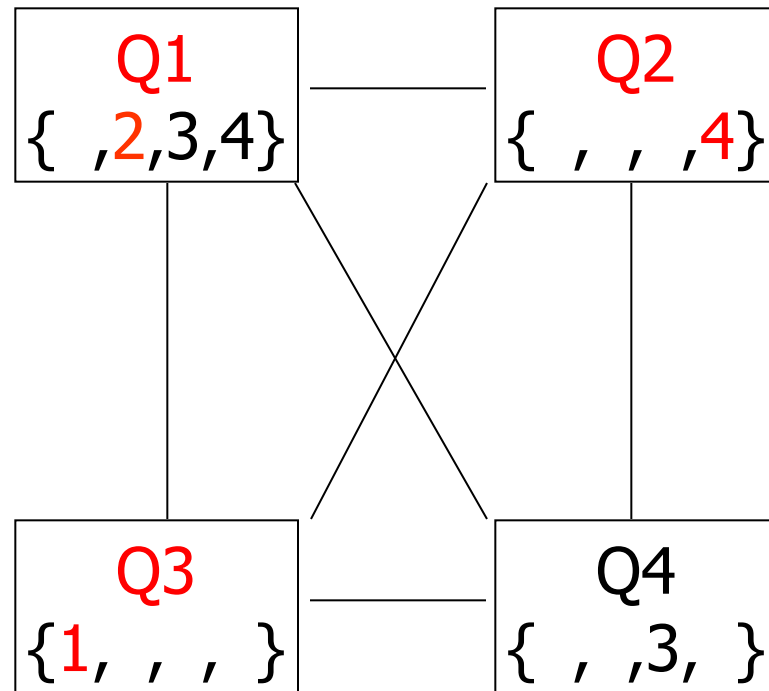
	1	2	3	4
1				
2				
3				
4				





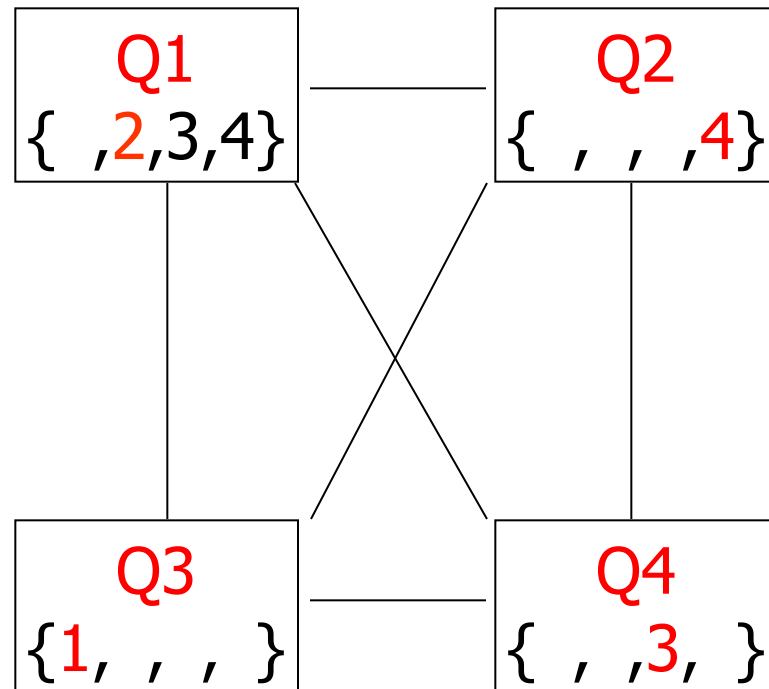
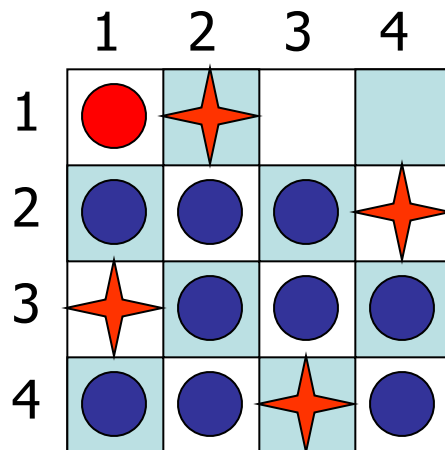
# 4-Queens Problem

	1	2	3	4
1				
2				
3				
4				



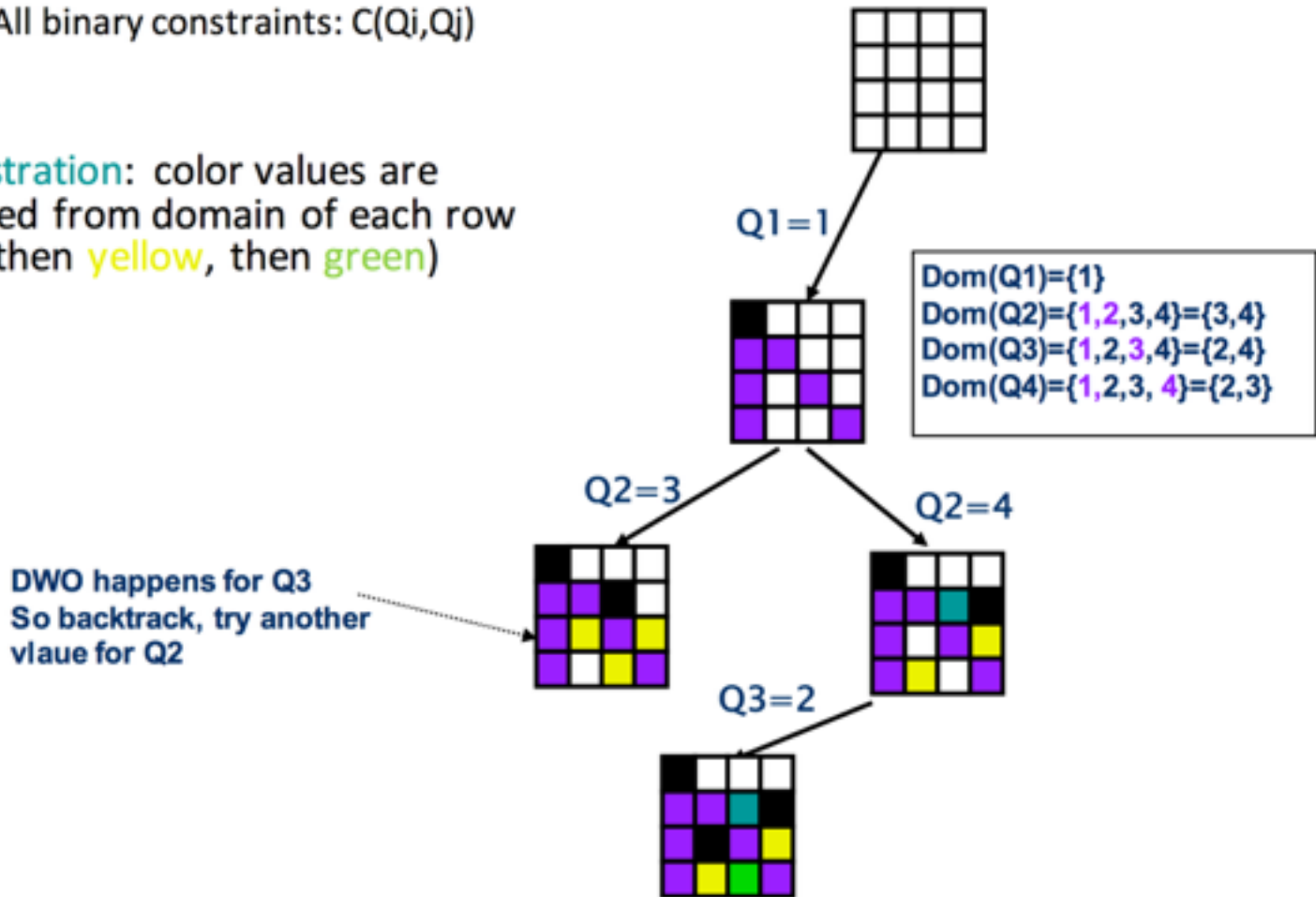
# 4-Queens Problem

We have now find a solution: an assignment of all variables to values of their domain so that all constraints are satisfied



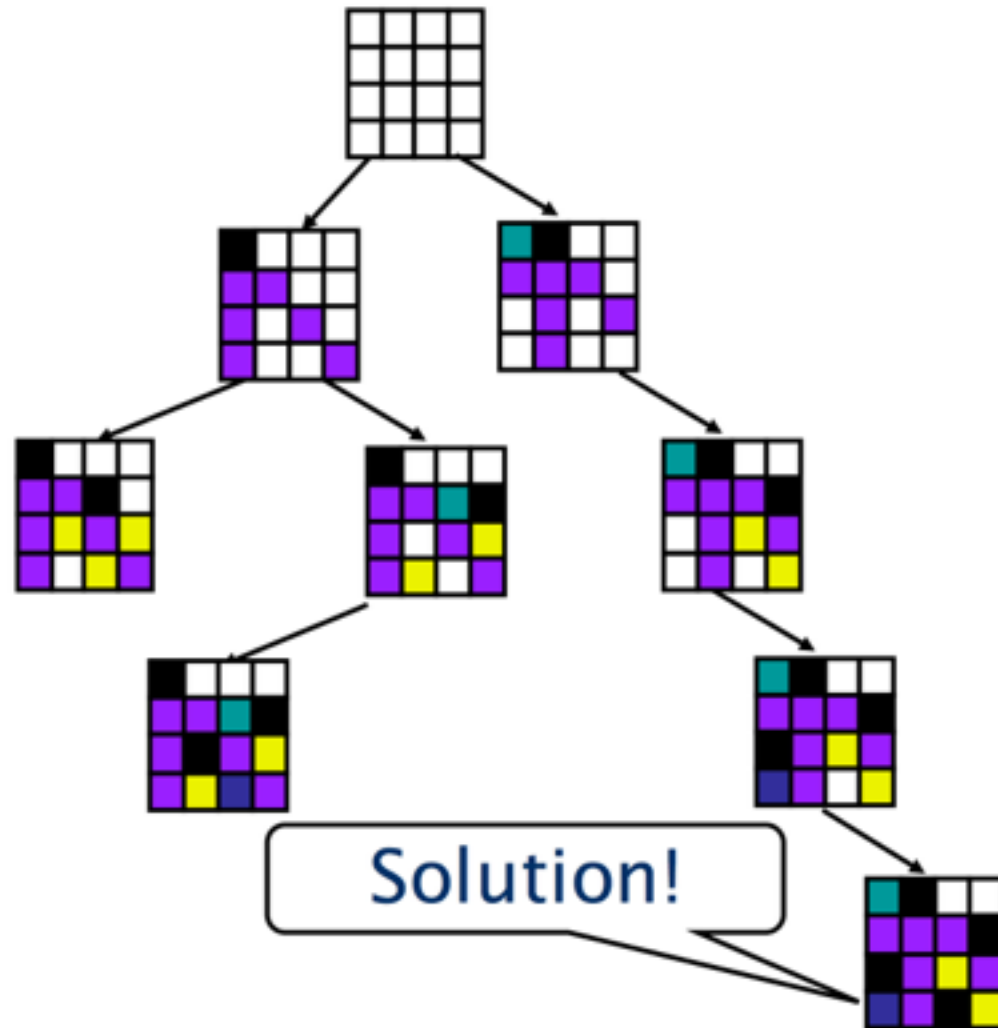
# Example: N-Queens FC search Space

- 4X4 Queens
  - $Q_1, Q_2, Q_3, Q_4$  with domain  $\{1..4\}$
  - All binary constraints:  $C(Q_i, Q_j)$
- **FC illustration:** color values are removed from domain of each row (blue, then yellow, then green)



## Example: N-Queens FC search Space

- 4X4 Queens  
continue...



# Example: Map Colouring

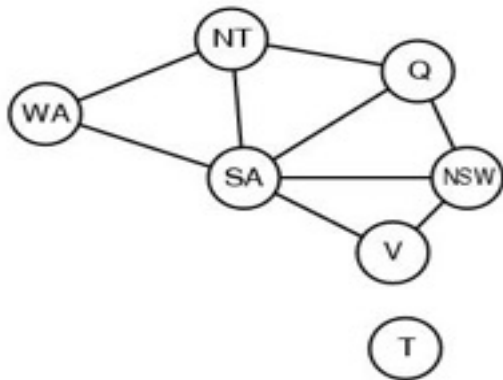
---

Colour the following map using *red*, *green*, and *blue* such that adjacent regions have different colours.



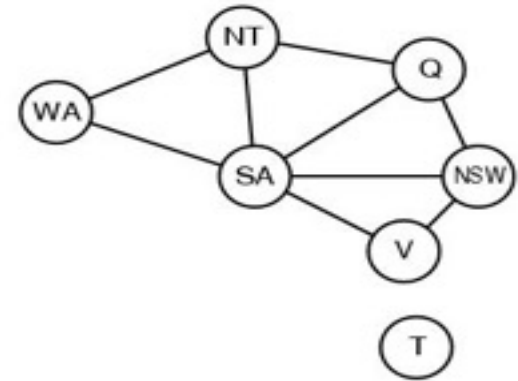
# Example: Map Colouring

- Model
  - **Variables:**  $WA, NT, Q, NSW, V, SA, T$
  - **Domains:**  $D_i = \{red, green, blue\}$
  - **Constraints:** adjacent regions must have different colors.
    - E.g.  $WA \neq NT$



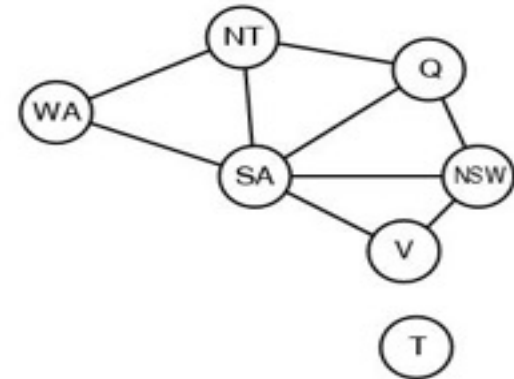
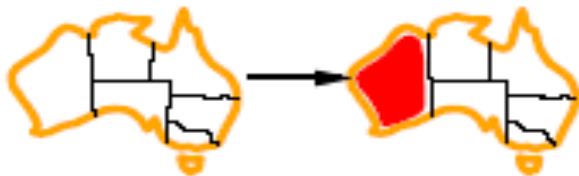
# Example: Map Colouring

- *Forward checking idea*: keep track of remaining legal values for unassigned variables.
- Terminate search when any variable has no legal values.



# Example: Map Colouring

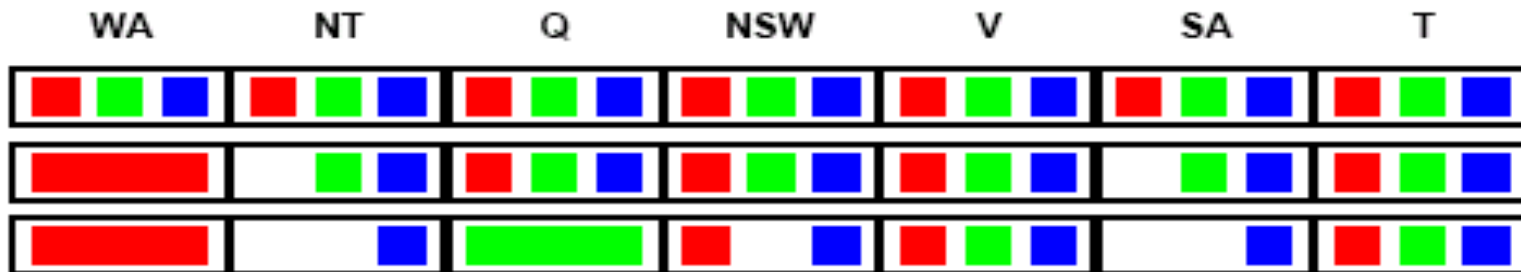
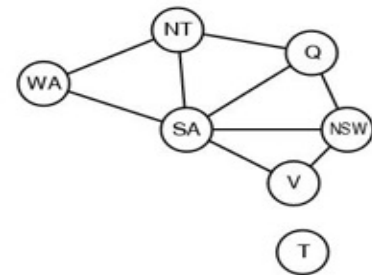
- Assign  $\{WA=red\}$
- Effects on other variables connected by constraints to WA
  - *NT can no longer be red*
  - *SA can no longer be red*





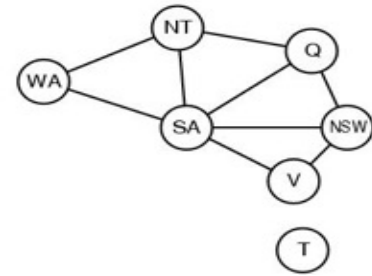
# Example: Map Colouring

- Assign  $\{Q=\text{green}\}$
- Effects on other variables connected by constraints with Q
  - *NT can no longer be green*
  - *NSW can no longer be green*
  - *SA can no longer be green*



# Example: Map Colouring

- Assign  $\{V=blue\}$
- Effects on other variables connected by constraints with V
  - NSW can no longer be blue
  - SA is empty
- FC has detected that partial assignment is *inconsistent* with the constraints and backtracking can occur.



WA	NT	Q	NSW	V	SA	T
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	

# FC: Restoring Values

---

- After we backtrack from the current assignment (in the for loop) we must restore the values that were pruned as a result of that assignment.
- Some bookkeeping needs to be done, as we must remember which values were pruned by which assignment (FCCheck is called at every recursive invocation of FC).

# Constraint Propagation

---

- Another form of propagation is to make each arc in a constraint graph **consistent**
- $C(X,Y)$  is **consistent** iff for every value of  $X$  there is some value of  $Y$  that satisfies  $C$ .
- We can use **inconsistencies** to remove values from the domains of variables:
  - e.g.  $C(X,Y): X > Y$   $\text{Dom}(X) = \{1,5,11\}$   $\text{Dom}(Y) = \{3,8,15\}$
  - for  $X = 1$  there is no value of  $Y$  s.t.  $Y$  is less than  $X$ . Remove 1 from the Domain of  $X$
  - for  $Y = 15$  there is no value of  $X$  s.t.  $X$  is more than  $Y$ . Remove 15 from the Domain of  $Y$
  - We obtain  $\text{Dom}(X) = \{5,11\}$  and  $\text{Dom}(Y) = \{3,8\}$ .
- Removing a value from a domain may trigger further inconsistency, so we have to repeat the procedure until everything is consistent.
- For efficient implementation, we keep track of inconsistent arcs by putting them in a Queue (See AC3 algorithm in the book).
- *This is stronger than forward checking. Why?*
- Checking for consistency can be done as a pre-processing step, or it can be directly integrated into a search algorithm (as we will see).

## Constraint Propagation: Generalized Arc Consistency (GAC)

---

$C(V_1, V_2, V_3, \dots, V_n)$  is GAC with respect to variable  $V_i$ , if and only if

For every value of  $V_i$ , there exist values of  $V_1, V_2, V_{i-1}, V_{i+1}, \dots, V_n$  that satisfy  $C$ .

Note that all combinations do not need to satisfy  $C$ , but for every value of  $V_i$ , some satisfying combination must be possible.

Also, note that values are removed from variable domains during search. So a constraint that is GAC with respect to a variable may become non-GAC with respect to that variable.

# Constraint Propagation: Generalized Arc Consistency (GAC)

---

$C(V_1, V_2, V_3, \dots, V_n)$  is GAC if and only if

It is GAC with respect to every variable in its scope.

A CSP is GAC if and only if

all of its constraints are GAC.

# Constraint Propagation: Arc Consistency

---

- Say we find a value  $d$  of variable  $V_i$  that is not consistent w.r.t. a constraint: that is, there is no assignments to the other variables that satisfy the constraint when  $V_i = d$ 
  - $d$  is said to be **Arc Inconsistent**
  - We **can remove  $d$**  from the domain of  $V_i$  as this value cannot lead to a solution (much like Forward Checking, but more powerful).
- Remember the example of  $C(X,Y): X > Y$   $\text{Dom}(X) = \{1, 5, 11\}$   $\text{Dom}(Y) = \{3, 8, 15\}$ 
  - **Arc Inconsistent values were pruned from domains.**

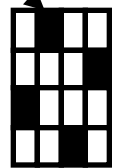
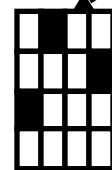
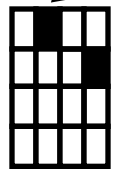
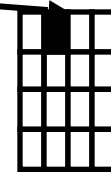
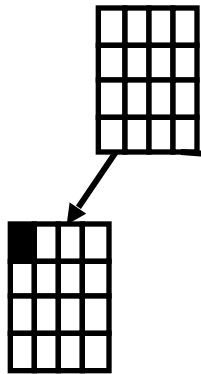
# Constraint Propagation: Arc Consistency

---

- If we apply arc consistency propagation during search the search tree's size will typically be much reduced in size.
- Removing a value from a variable domain may trigger further inconsistency, so we have to repeat the procedure until everything is consistent.
  - We put constraints on a queue and add new constraints to the queue as we need to check for arc consistency.



# Example: N-Queens GAC search Space

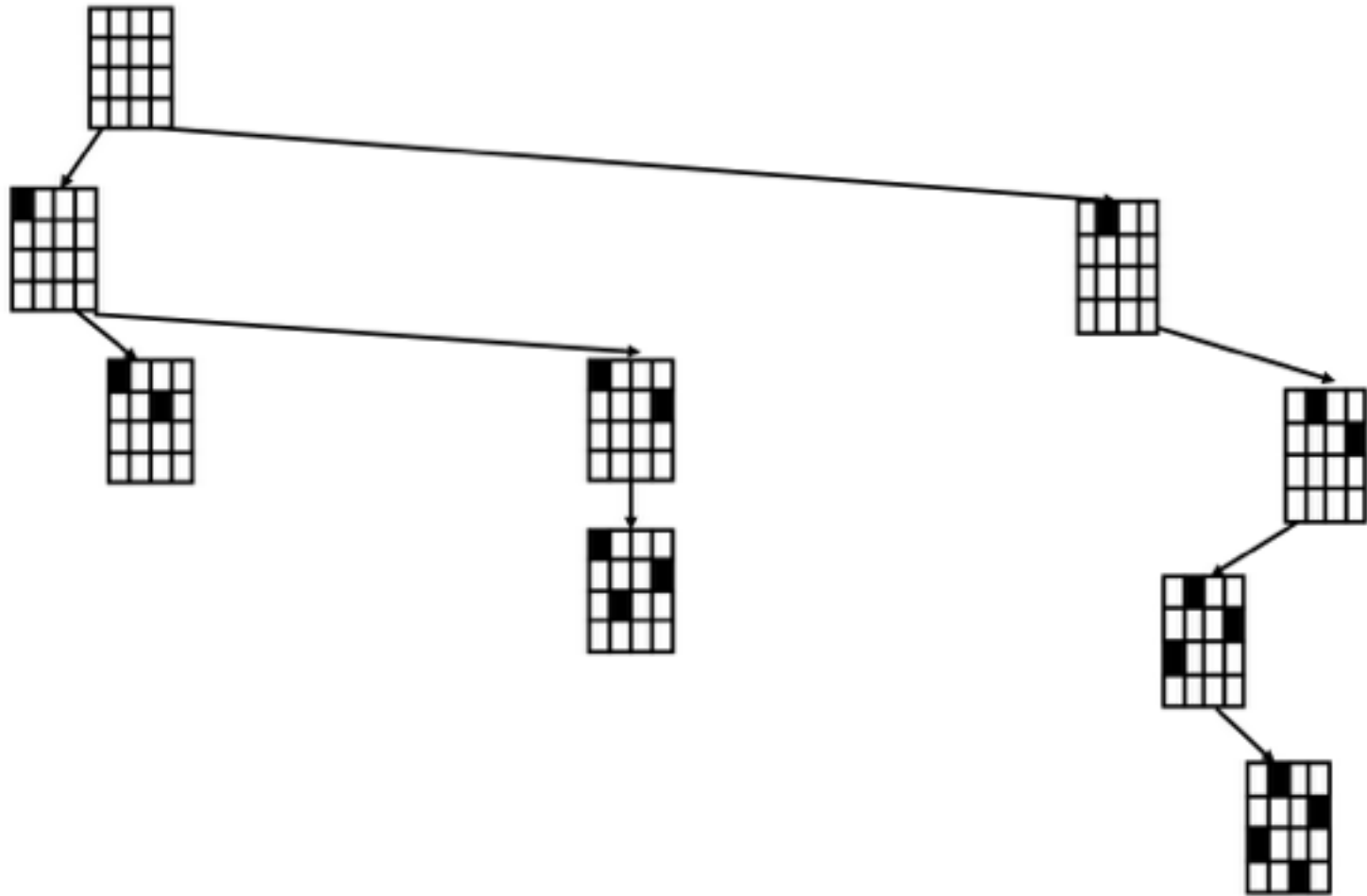


arc consistency stages:

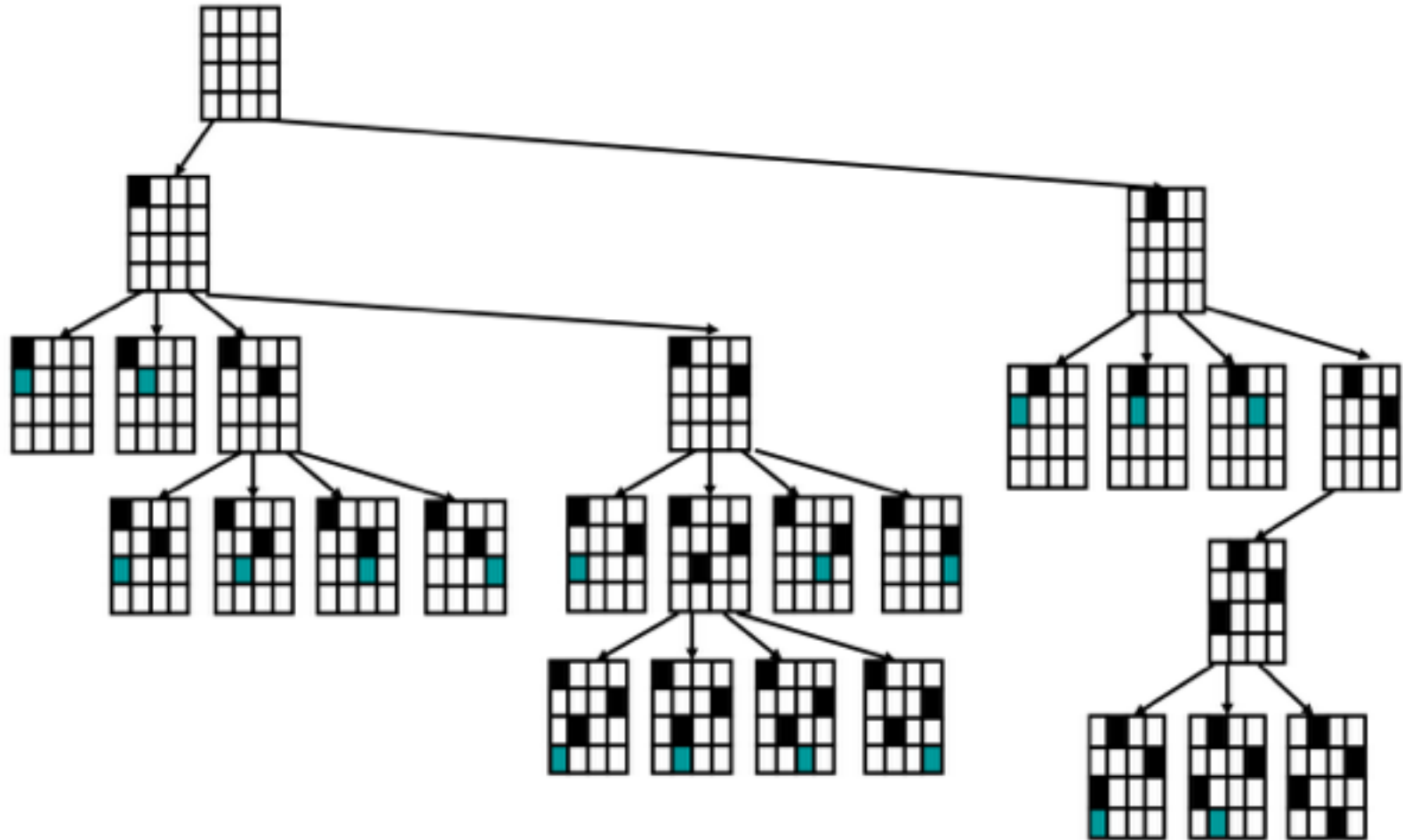
1.  $V2 = \{3, 4\}$ ,  $V3 = \{2, 4\}$ ,  $V4 = \{2, 3\}$   
( $V2=1, 2$  &  $V3 = 1, 3$  &  $V3 = 1, 4$  are inconsistent with  $V1=1$ )
2.  $V2 = \{4\}$  ( $V2=3$  is inconsistent with both values in  $\text{CurDom}[V3]$ )
3.  $V3 = \{2\}$  ( $V3 = 4$  is inconsistent with values in  $\text{CurDom}[V2]$ )
4.  $V4 = \{\}$  (both values for  $V4$  inconsistent with values in  $\text{CurDom}[V3]$ )

DWO

# Example: N-Queens FC search Space



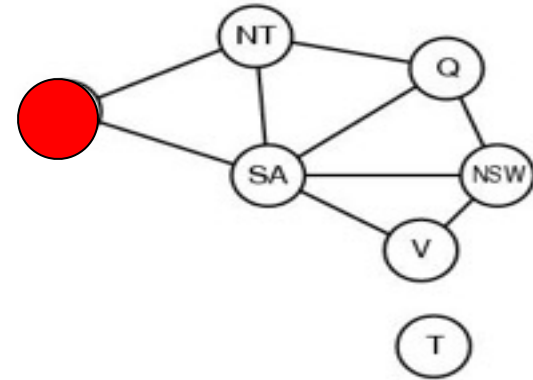
# Example: N-Queens BT search Space



# Example: Map Colouring

---

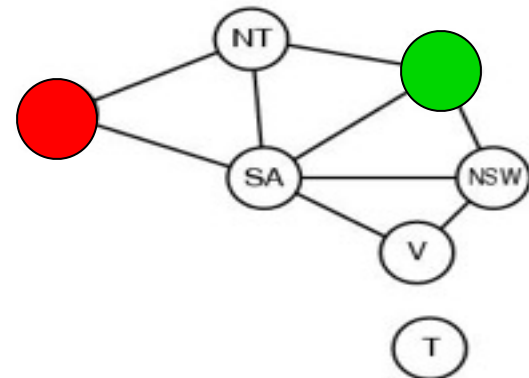
- Assign  $\{WA=red\}$
- Effects on other variables connected by constraints to WA
  - *NT can no longer be red =  $\{G, B\}$*
  - *SA can no longer be red =  $\{G, B\}$*
- *All other values are arc-consistent*



# Example - Map Colouring

- Assign  $\{Q=green\}$
- Effects on other variables connected by constraints with Q
  - *NT can no longer be green* =  $\{B\}$
  - *NSW can no longer be green* =  $\{R, B\}$
  - *SA can no longer be green* =  $\{B\}$
- *DWO there is no value for SA that will be consistent with  $NT \neq SA$  and  $NT = B$*

*Note Forward Checking would not have detected this DWO.*

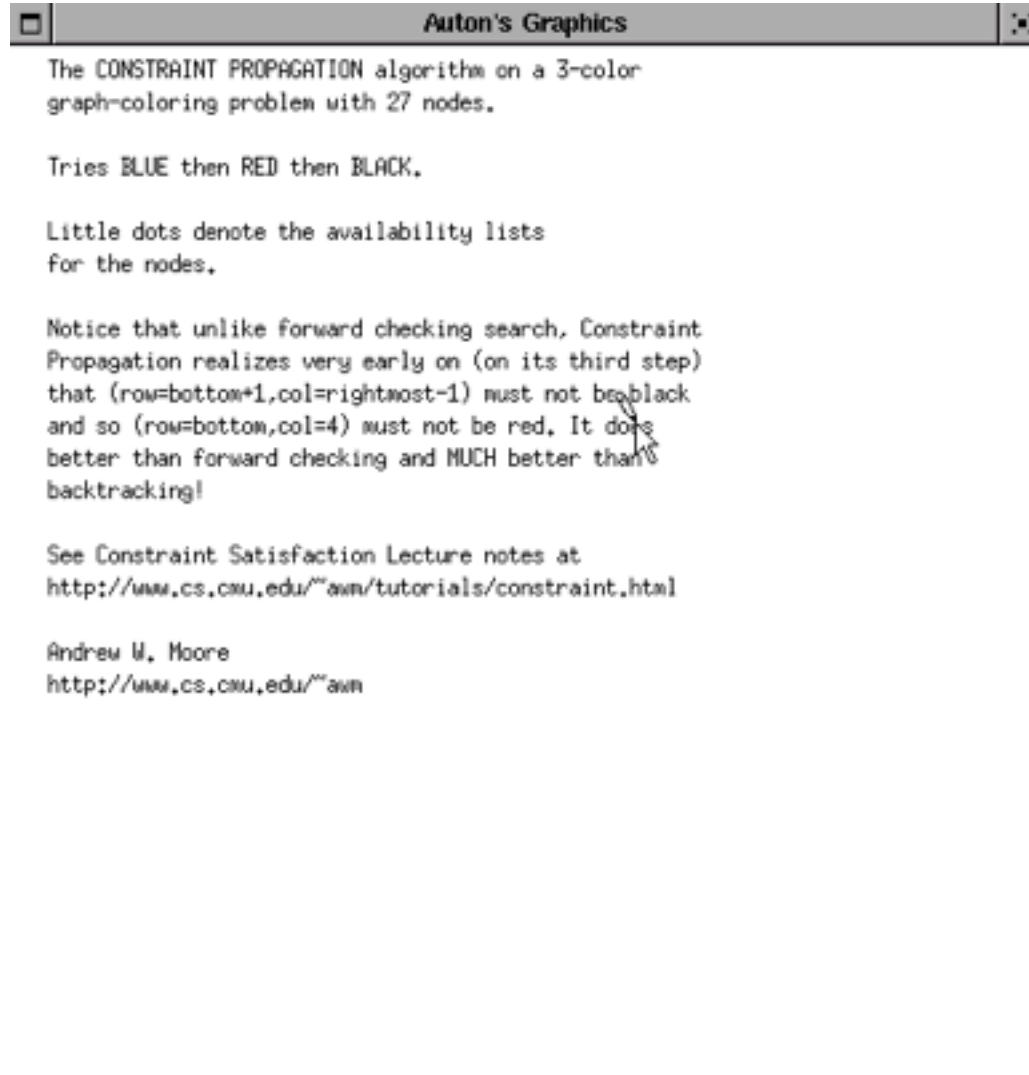


# GAC Algorithm

---

- We make all constraints GAC at every node of the search space.
- This is accomplished by removing from the domains of the variables all arc inconsistent values.

# Constrain Propagation Example



# GAC Algorithm, enforce GAC during search

---

```
GAC(Level) /*Maintain GAC Algorithm */
    If all variables are assigned
        PRINT Value of each Variable
        RETURN or EXIT (RETURN for more solutions)
                        (EXIT for only one solution)
    V := PickAnUnassignedVariable()
    Assigned[V] := TRUE
    for d := each member of CurDom(V)
        Value[V] := d
        Prune all values of V  $\neq$  d from CurDom[V]
        for each constraint C whose scope contains V
            Put C on GACQueue
        if (GAC_Enforce() != DWO)
            GAC(Level+1) /*all constraints were ok*/
        RestoreAllValuesPrunedFromCurDoms()
    Assigned[V] := FALSE
    return;
```



# Enforce GAC (prune all GAC inconsistent)

---

GAC\_Enforce()

```
// GAC-Queue contains all constraints one of whose variables has  
// had its domain reduced. At the root of the search tree  
// first we run GAC_Enforce with all constraints on GAC-Queue
```

```
while GACQueue not empty
```

```
    C = GACQueue.extract()
```

```
    for V := each member of scope(C)
```

```
        for d := CurDom[V]
```

```
            Find an assignment A for all other  
            variables in scope(C) such that  
            C(A  $\cup$  V=d) = True
```

```
            if A not found
```

```
                CurDom[V] = CurDom[V] - d
```

```
                if CurDom[V] =  $\emptyset$ 
```

```
                    empty GACQueue
```

```
                    return DWO //return immediately
```

```
                else
```

```
                    push all constraints C' such that  
                    V  $\in$  scope(C') and C'  $\notin$  GACQueue  
                    on to GACQueue
```

```
return TRUE //while loop exited without DWO
```

# Enforce GAC

---

- A **support** for  $V=d$  in constraint  $C$  is an assignment  $A$  to all of the other variables in  $\text{scope}(C)$  such that  $A \cup \{V=d\}$  satisfies  $C$ . ( $A$  is what the algorithm's inner loop looks for).
- Smarter implementations **keep track of “supports”** to avoid having to search through all possible assignments to the other variables for a satisfying assignment.

# Enforce GAC

---

- Rather than search for a satisfying assignment to  $C$  containing  $V=d$ , we check to see if the current support is still valid: i.e., all values it assigns still lie in the variable's current domains
- Also we take advantage that a support for  $V=d$ , e.g.  $\{V=d, X=a, Y=b, Z=c\}$  is also a support for  $X=a$ ,  $Y=b$ , and  $Z=c$

# Enforce GAC

---

- However, finding a support for  $V=x$  in constraint  $C$  still in the worst case requires  $O(d^k)$  work, where  $k$  is the arity of  $C$ , i.e.,  $|\text{scope}(C)|$  and  $d$  the size of the domain.
- Note that DFS would require  $O(d^N)$  time, where  $N$  is the number of all variables to be assigned.
- This tell us that GAC can be much faster, assuming constraints with relatively small arity
- Another key development in practice is that for some constraints this computation can be done in polynomial time.  
E.g., all-diff( $V_1, \dots, V_n$ ) we can be check if  $V_i=d$  has a support in the current domains of the other variables in polynomial time using ideas from graph theory. We do not need to examine all combinations of values for the other variables looking for a support.

# GAC enforce example

$GAC(C_{SS8}) \rightarrow \text{CurDom of } V_{1,5}, V_{1,6}, V_{2,4}, V_{3,4}, V_{3,6} = \{1, 2, 3, 4, 8\}$

$GAC(C_{R1}) \rightarrow \text{CurDom of } V_{1,7}, V_{1,8} = \{2, 3, 7, 9\}$

$\text{CurDom of } V_{1,5}, V_{1,6} = \{2, 3\}$

$GAC(C_{SS8}) \rightarrow \text{CurDom of } V_{2,4}, V_{3,4}, V_{3,6} = \{1, 4, 8\}$

$GAC(C_{C5}) \rightarrow \text{CurDom of } V_{5,5}, V_{9,5} = \{2, 6, 8\}$

$\rightarrow \text{CurDom of } V_{1,5} = \{2\}$

$GAC(C_{SS8}) \rightarrow \text{CurDom of } V_{1,6} = \{3\}$

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

= All-diff

$C_{SS2} = \text{All-diff}(V_{1,4}, V_{1,5}, V_{1,6}, V_{2,4}, V_{2,5}, V_{2,6}, V_{3,4}, V_{3,5}, V_{3,6})$

$C_{R1} = \text{All-diff}(V_{1,1}, V_{1,2}, V_{1,3}, V_{1,4}, V_{1,5}, V_{1,6}, V_{1,7}, V_{1,8}, V_{1,9})$

$C_{C5} = \text{All-diff}(V_{1,5}, V_{2,5}, V_{3,5}, V_{4,5}, V_{5,5}, V_{6,5}, V_{7,5}, V_{8,5}, V_{9,5})$

By going back and forth between constraints we get more values pruned.

# Variable and Value Ordering Heuristics

---

- Heuristics can be used to determine
  - the order in which variables are assigned:  
**PickUnassignedVariable()**
  - the order of values tried for each variable.
- The choice of the next variable can vary from branch to branch, e.g.,
  - under the assignment  $V1=a$  we might choose to assign  $V4$  next, while under  $V1=b$  we might choose to assign  $V5$  next.
- This “**dynamically**” chosen variable ordering has a tremendous impact on performance.

# Minimum Remaining Values Heuristic (MRV)

---

FC gives us for free a very powerful heuristic to guide us which variables to try next:

- Always branch on a variable with **the smallest remaining values** (smallest CurDom).
- If a variable has only one value left, that value is forced, so we should propagate its consequences immediately.
- This heuristic tends to produce skinny trees at the top. This means that more variables can be instantiated with fewer nodes searched, and thus more constraint propagation/DWO failures occur when the tree starts to branch out (we start selecting variables with larger domains)
- We can find a inconsistency much faster

# MRV Heuristic: Human Analogy

- What variables would you try first?

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

Domain of each variable:  
 $\{1, \dots, 9\}$

(1, 5): impossible values:

Row:  $\{1, 4, 5, 6, 8\}$

Column:  $\{1, 3, 4, 5, 7, 9\}$

Subsquare:  $\{5, 7, 9\}$

→ Domain =  $\{2\}$

(9, 5): impossible values:

Row:  $\{1, 5, 7, 8, 9\}$

Column:  $\{1, 3, 4, 5, 7, 9\}$

Subsquare:  $\{1, 5, 7, 9\}$

→ Domain =  $\{2, 6\}$

**Most restricted variables! = MRV**

After assigning value 2 to  
cell (1,5): Domain =  $\{6\}$

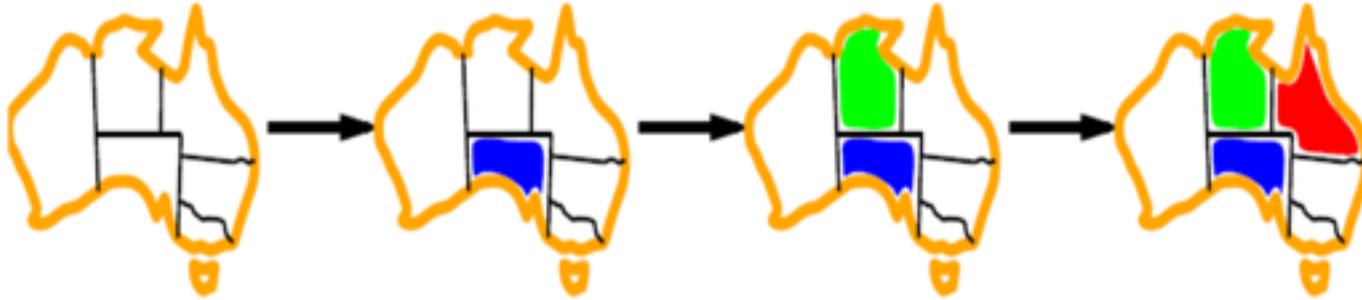


# MRV Empirically

---

- FC often is about 100 times faster than BT
- FC with MRV (minimal remaining values) often 10000 times faster.
- But on some problems the speed up can be much greater
  - Converts problems that are not solvable to problems that are solvable.
- Still FC is not that powerful. Other more powerful forms of constraint propagation are used in practice.
- Try the map colouring example with MRV.

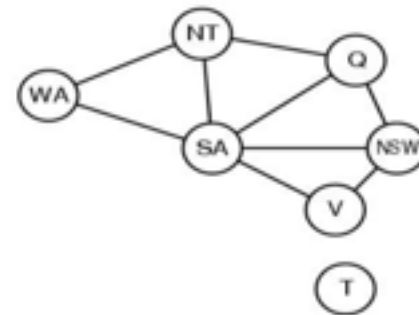
# Degree Heuristic



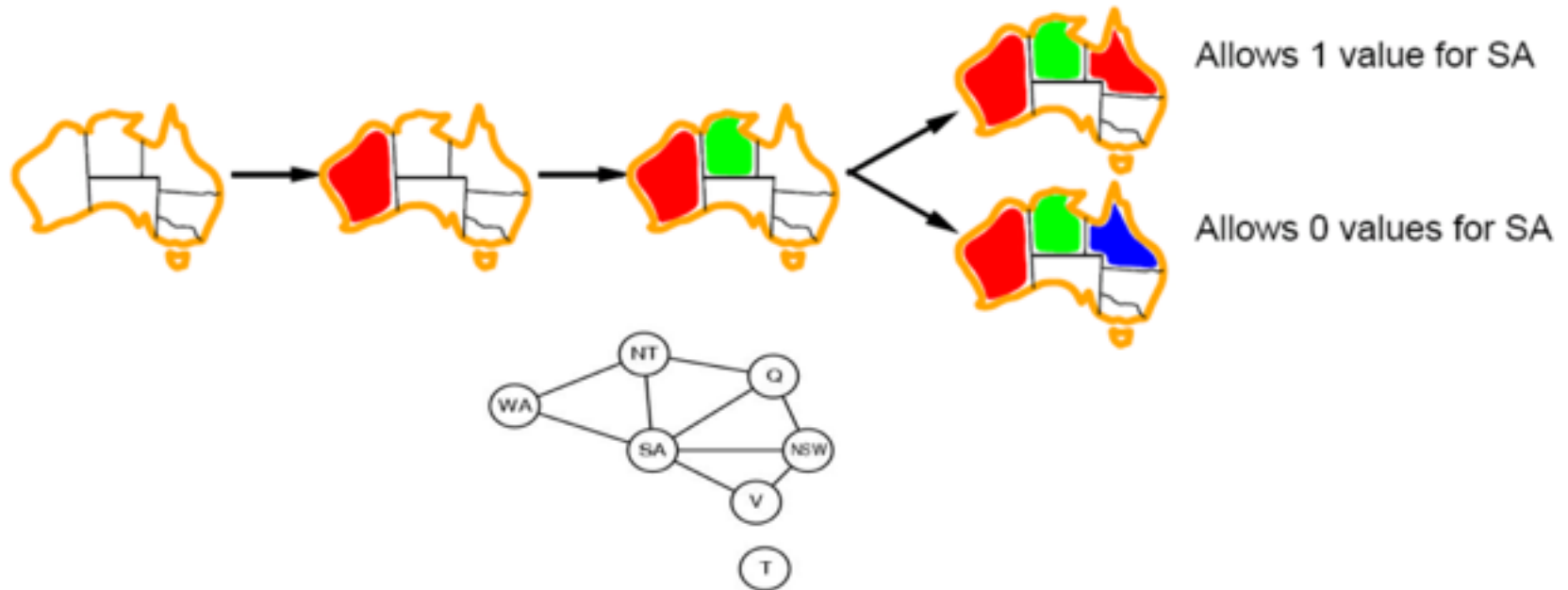
*The Heuristic Rule:* select variable that is involved in the largest number of constraints on other unassigned variables.

Degree heuristic can be useful as a tie breaker.

*In what order should a variable's values be tried?*



# Least Constraining Value Heuristic



*The Heuristic Rule:* given a variable choose the least constraining value

- leaves the maximum flexibility for subsequent variable assignments

# Local Search

---

- As we've discussed, CSPs problems often don't require preserve the path to the goal (e.g. scheduling problems or network optimizations).
- **Local search algorithms** can also be useful here. These operate by starting at a single current state and moving to neighbours of that state.
  - They need **an objective function that provides the value of each state**. Goals will have the highest (or lowest) values (i.e. a global maximum or minimum).
- **Algorithms include:**
  - **Hill Climbing** tries to move to a neighbour with the highest value. There is danger of being stuck in a local maximum. **So some randomness is added** to “shake” the search out of local maxima.
  - **Simulated Annealing** introduces some randomness. Take a random move and if it **improves the situation** always accept it, otherwise accept it with a probability  $<1$ .
- [If interested read these two algorithms from the R&N book].

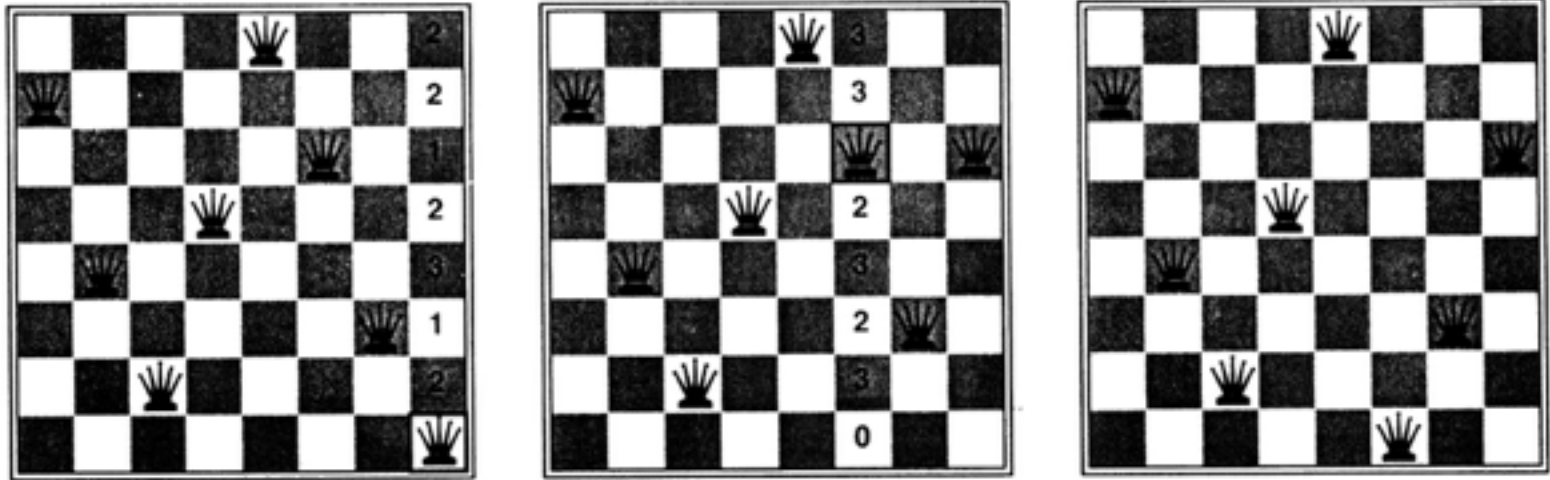
# Local Search

---

## Min-conflicts algorithm:

- The objective function (or heuristic) is given by the number of conflicts between variables
- Search for states that result in a minimum number of conflicts between variables

# Local Search



At each stage, a queen is chosen for reassignment in its column.

# conflicts (# attacking queens) is shown in each square.

The algorithm moves the queen to the min-conflict square, breaking ties randomly.

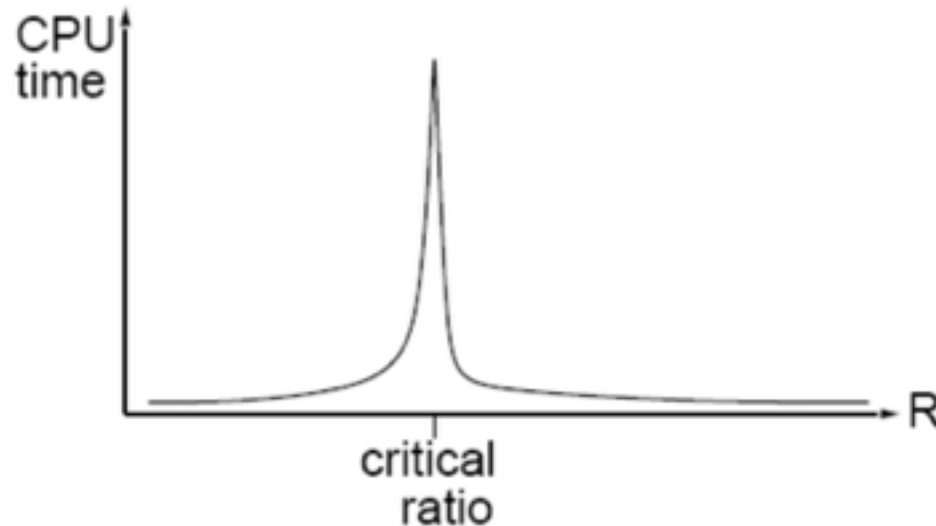
Demo of min-conflicts on a graph colouring problem: <https://www.youtube.com/watch?v=QGH9g-CNPxQ>

# Min-Conflicts performance

---

Given a random initial state, the min-conflicts algorithm solves n-queens **in almost constant time** for arbitrary n.

$$R = (\# \text{ of constraints})/(\# \text{ of variables})$$



# Other real-world applications of CSP

---

- Assignment problems
  - who teaches what class
- Timetabling problems
  - exam schedule
- Transportation scheduling
- Floor planning
- Factory scheduling
- Hardware configuration
  - a set of compatible components

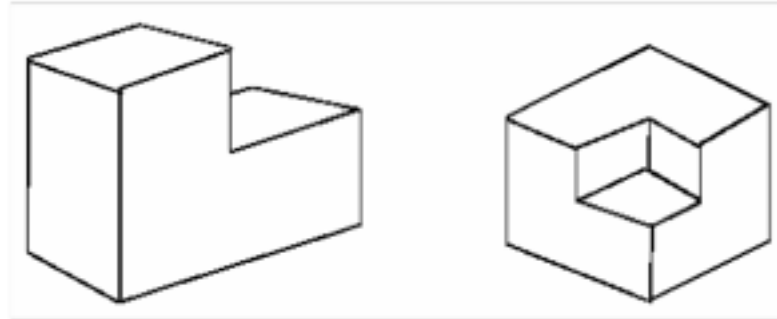


# Other Interesting CSPs

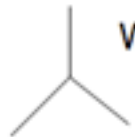
---

## The Waltz algorithm

One of the earliest examples of a computation posed as a CSP.  
The Waltz algorithm is for interpreting line drawings of solid polyhedra.



Look at all intersections.



What kind of intersection could this be? A concave intersection of three faces? Or an external convex intersection?

Adjacent intersections impose constraints on each other. Use CSP to find a unique set of labelings. Important step to “understanding” the image.

SI

# Other Interesting CSPs

---

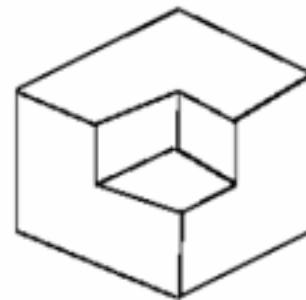
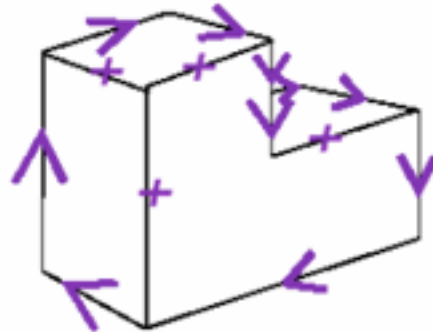
## Waltz Alg. on simple scenes

Assume all objects:

- Have no shadows or cracks
- Three-faced vertices
- "General position": no junctions change with small movements of the eye.

Then each line on image is one of the following:

- Boundary line (edge of an object) (<) with right hand of arrow denoting "solid" and left hand denoting "space"
- Interior convex edge (+)
- Interior concave edge (-)



# Other Interesting CSPs

## 18 legal kinds of junctions



Given a representation of the diagram, label each junction in one of the above manners.

The junctions must be labeled so that lines are labeled consistently at both ends.

Can you formulate that as a CSP? *FUN FACT: Constraint Propagation always works perfectly.*

# Other Interesting CSPs

