Constraint Satisfaction Problems (CSP)

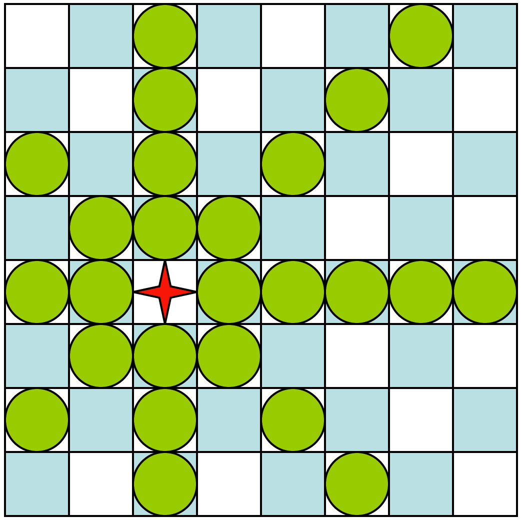
(Where we postpone making difficult decisions until they become easy to make)

R&N: Chap. 6

What we will try to do ...

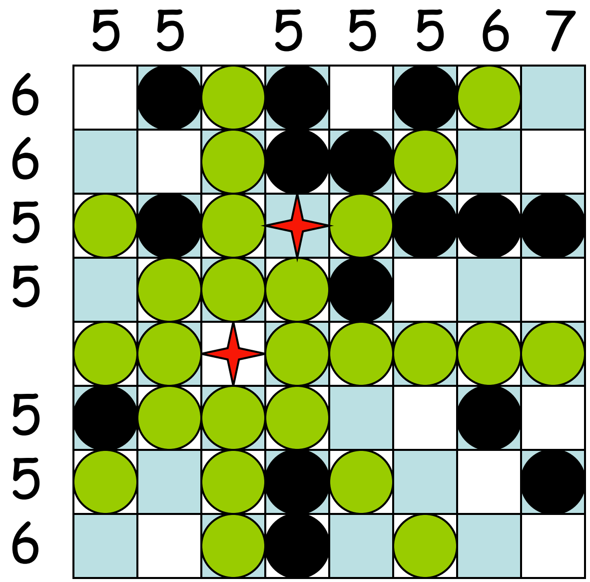
- Search techniques make choices in an often arbitrary order. Often little information is available to make each of them
- In many problems, the same states can be reached independent of the order in which choices are made ("commutative" actions)
- Can we solve such problems more efficiently by picking the order appropriately? Can we even avoid making any choice?

Constraint Propagation



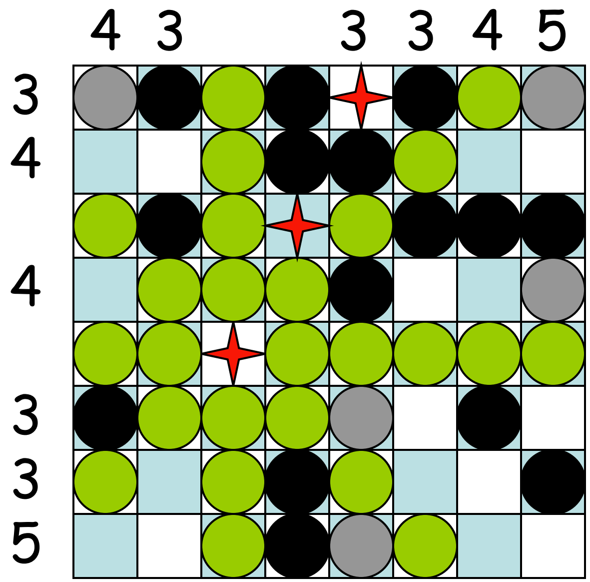
- Place a queen in a square
- Remove the attacked squares from future consideration

Constraint Propagation



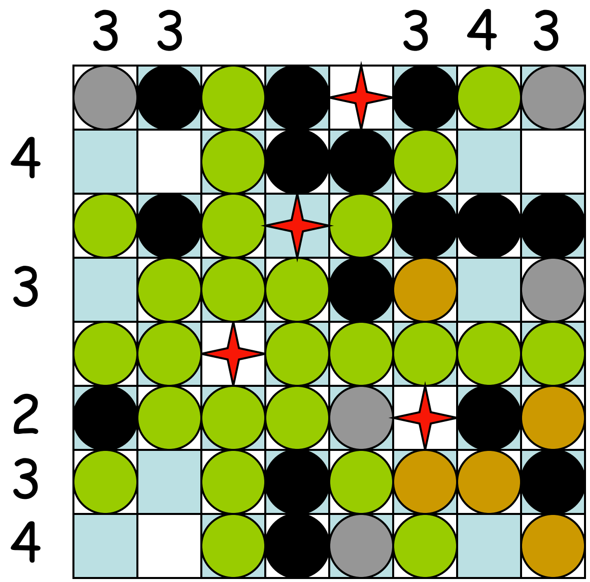
- Count the number of non-attacked squares in every row and column
- Place a queen in a row or column with minimum number
- Remove the attacked squares from future consideration

Constraint Propagation

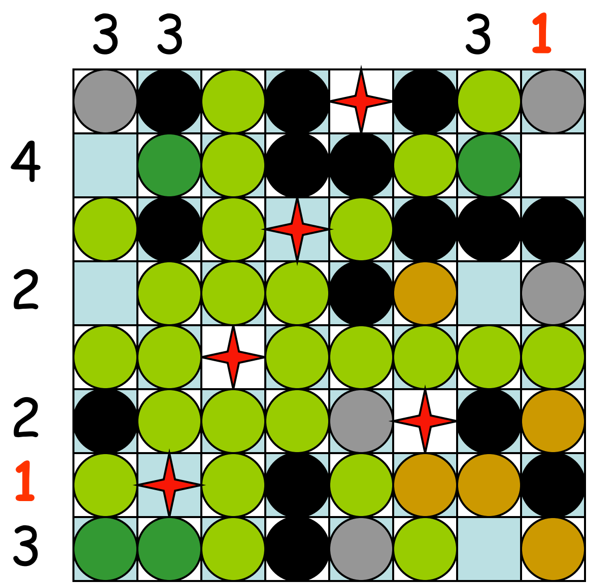


- Repeat

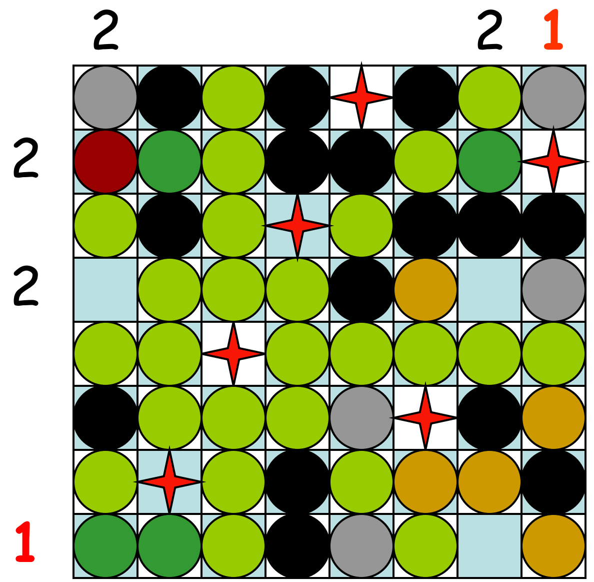
Constraint Propagation



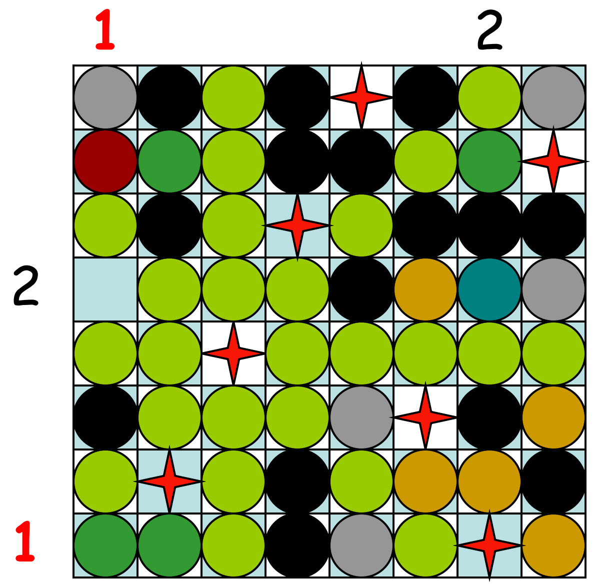
Constraint Propagation



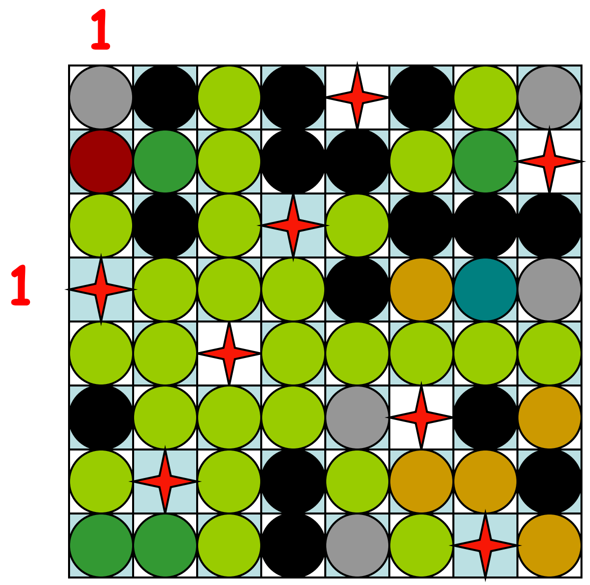
Constraint Propagation



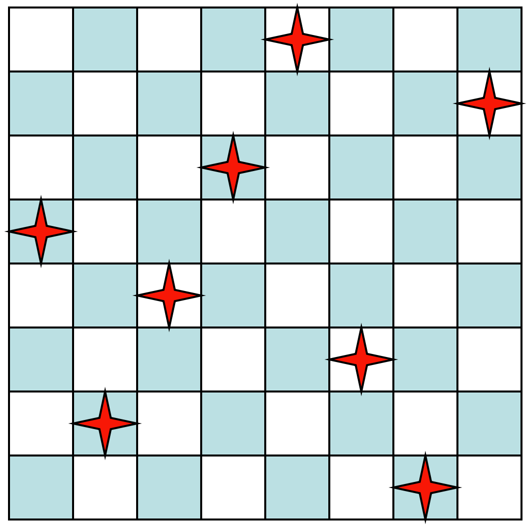
Constraint Propagation



Constraint Propagation



Constraint Propagation



What do we need?

- More than just a successor function and a goal test
- We also need:
  - A means to propagate the constraints imposed by one queen's position on the positions of the other queens
  - An early failure test

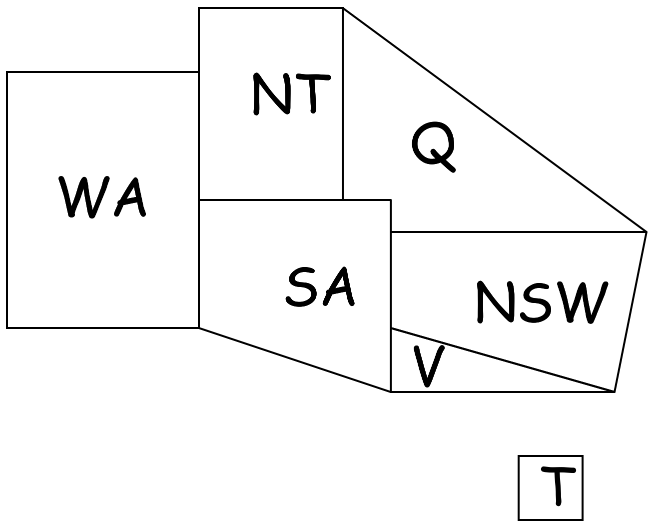
→ Explicit representation of constraints

→ Constraint propagation algorithms

Constraint Satisfaction Problem (CSP)

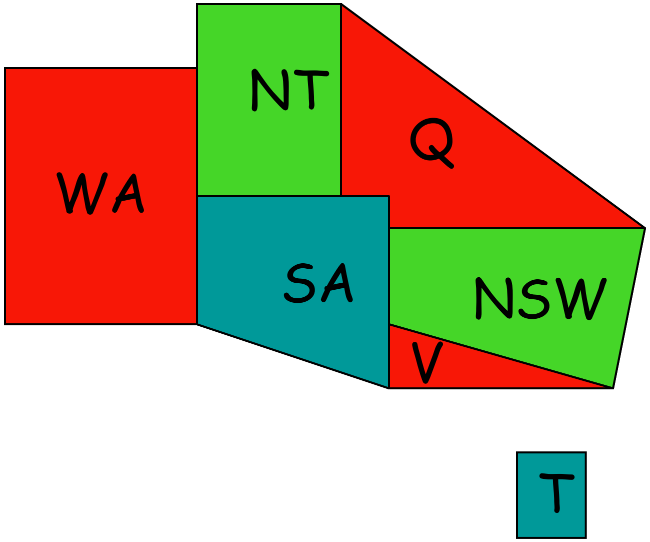
- Set of **variables** $\{X_1, X_2, ..., X_n\}$
- Each variable X_i has a **domain** D_i of possible values. Usually, D_i is finite
- Set of **constraints** $\{C_1, C_2, ..., C_p\}$
- Each constraint relates a subset of variables by specifying the valid combinations of their values
- Goal: Assign a value to every variable such that all constraints are satisfied

Map Coloring



- 7 variables {WA,NT,SA,Q,NSW,V,T}
- Each variable has the same domain:
  {red, green, blue}
- No two adjacent variables have the same value:
  WA≠NT, WA≠SA, NT≠SA, NT≠Q, SA≠Q,
  SA≠NSW, SA≠V, Q≠NSW, NSW≠V

Map Coloring



- 7 variables {WA,NT,SA,Q,NSW,V,T}
- Each variable has the same domain:
  {red, green, blue}
- No two adjacent variables have the same value:
  WA≠NT, WA≠SA, NT≠SA, NT≠Q, SA≠Q,
  SA≠NSW, SA≠V, Q≠NSW, NSW≠V

8-Queen Problem

- 8 variables X_i, $i = 1$ to 8
- The domain of each variable is: {1,2,…,8}
- Constraints are of the forms:
  - $X_i = k \rightarrow X_j \neq k$ for all $j = 1$ to 8, $j \neq i$
  - Similar constraints for diagonals

All constraints are binary

Street Puzzle

1	2	3	4	5

N_i = {English, Spaniard, Japanese, Italian, Norwegian}
C_i = {Red, Green, White, Yellow, Blue}
D_i = {Tea, Coffee, Milk, Fruit-juice, Water}
J_i = {Painter, Sculptor, Diplomat, Violinist, Doctor}
A_i = {Dog, Snails, Fox, Horse, Zebra}

The Englishman lives in the Red house
The Spaniard has a Dog
The Japanese is a Painter
The Italian drinks Tea
The Norwegian lives in the first house on the left
The owner of the Green house drinks Coffee
The Green house is on the right of the White house
The Sculptor breeds Snails
The Diplomat lives in the Yellow house
The owner of the middle house drinks Milk
The Norwegian lives next door to the Blue house
The Violinist drinks Fruit juice
The Fox is in the house next to the Doctor's
The Horse is next to the Diplomat's

Who owns the Zebra?
Who drinks Water?

Street Puzzle

1	2	3	4	5

N_i = {English, Spaniard, Japanese, Italian, Norwegian}
C_i = {Red, Green, White, Yellow, Blue}
D_i = {Tea, Coffee, Milk, Fruit-juice, Water}
J_i = {Painter, Sculptor, Diplomat, Violinist, Doctor}
A_i = {Dog, Snails, Fox, Horse, Zebra}

$\forall i,j \in [1,5],\ i \neq j,\ N_i \neq N_j$

$\forall i,j \in [1,5],\ i \neq j,\ C_i \neq C_j$

...

The Englishman lives in the Red house
The Spaniard has a Dog
The Japanese is a Painter
The Italian drinks Tea
The Norwegian lives in the first house on the left
The owner of the Green house drinks Coffee
The Green house is on the right of the White house
The Sculptor breeds Snails
The Diplomat lives in the Yellow house
The owner of the middle house drinks Milk
The Norwegian lives next door to the Blue house
The Violinist drinks Fruit juice
The Fox is in the house next to the Doctor's
The Horse is next to the Diplomat's

Street Puzzle

1	2	3	4	5

N_i = {English, Spaniard, Japanese, Italian, Norwegian}
C_i = {Red, Green, White, Yellow, Blue}
D_i = {Tea, Coffee, Milk, Fruit-juice, Water}
J_i = {Painter, Sculptor, Diplomat, Violinist, Doctor}
A_i = {Dog, Snails, Fox, Horse, Zebra}

The Englishman lives in the Red house → $(N_i = \text{English}) \Leftrightarrow (C_i = \text{Red})$
The Spaniard has a Dog
The Japanese is a Painter → $(N_i = \text{Japanese}) \Leftrightarrow (J_i = \text{Painter})$
The Italian drinks Tea
The Norwegian lives in the first house on the left → $(N_1 = \text{Norwegian})$
The owner of the Green house drinks Coffee
The Green house is on the right of the White house
The Sculptor breeds Snails
The Diplomat lives in the Yellow house
The owner of the middle house drinks Milk
The Norwegian lives next door to the Blue house
The Violinist drinks Fruit juice
The Fox is in the house next to the Doctor's
The Horse is next to the Diplomat's

$$\left\{\begin{array}{l} (C_i = \text{White}) \Leftrightarrow (C_{i+1} = \text{Green}) \\ (C_5 \neq \text{White}) \\ (C_1 \neq \text{Green}) \end{array}\right.$$

left as an exercise

Street Puzzle

1 2 3 4 5

N_i = {English, Spaniard, Japanese, Italian, Norwegian}
C_i = {Red, Green, White, Yellow, Blue}
D_i = {Tea, Coffee, Milk, Fruit-juice, Water}
J_i = {Painter, Sculptor, Diplomat, Violinist, Doctor}
A_i = {Dog, Snails, Fox, Horse, Zebra}

The Englishman lives in the Red house → $(N_i = \text{English}) \Leftrightarrow (C_i = \text{Red})$
The Spaniard has a Dog
The Japanese is a Painter → $(N_i = \text{Japanese}) \Leftrightarrow (J_i = \text{Painter})$
The Italian drinks Tea
The Norwegian lives in the first house on the left → $(N_1 = \text{Norwegian})$
The owner of the Green house drinks Coffee
The Green house is on the right of the White house
The Sculptor breeds Snails
The Diplomat lives in the Yellow house
The owner of the middle house drinks Milk
The Norwegian lives next door to the Blue house
The Violinist drinks Fruit juice
The Fox is in the house next to the Doctor's
The Horse is next to the Diplomat's

$$\left\{\begin{array}{l} (C_i = \text{White}) \Leftrightarrow (C_{i+1} = \text{Green}) \\ (C_5 \neq \text{White}) \\ (C_1 \neq \text{Green}) \end{array}\right.$$

unary constraints

Street Puzzle

1	2	3	4	5

N_i = {English, Spaniard, Japanese, Italian, Norwegian}
C_i = {Red, Green, White, Yellow, Blue}
D_i = {Tea, Coffee, Milk, Fruit-juice, Water}
J_i = {Painter, Sculptor, Diplomat, Violinist, Doctor}
A_i = {Dog, Snails, Fox, Horse, Zebra}

The Englishman lives in the Red house
The Spaniard has a Dog
The Japanese is a Painter
The Italian drinks Tea
The Norwegian lives in the first house on the left → N_1 = Norwegian
The owner of the Green house drinks Coffee
The Green house is on the right of the White house
The Sculptor breeds Snails
The Diplomat lives in the Yellow house
The owner of the middle house drinks Milk → D_3 = Milk
The Norwegian lives next door to the Blue house
The Violinist drinks Fruit juice
The Fox is in the house next to the Doctor's
The Horse is next to the Diplomat's

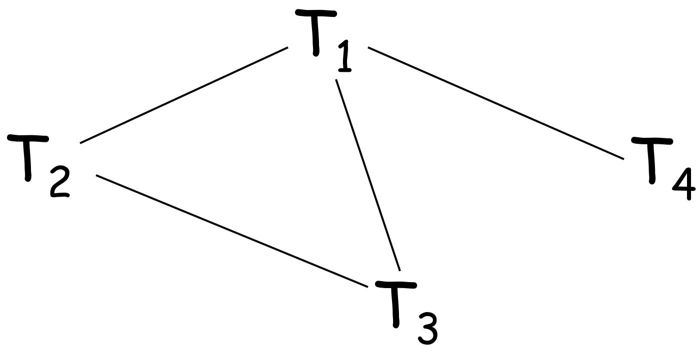
Street Puzzle

1	2	3	4	5

N_i = {English, Spaniard, Japanese, Italian, Norwegian}
C_i = {Red, Green, White, Yellow, Blue}
D_i = {Tea, Coffee, Milk, Fruit-juice, Water}
J_i = {Painter, Sculptor, Diplomat, Violinist, Doctor}
A_i = {Dog, Snails, Fox, Horse, Zebra}

The Englishman lives in the Red house → $C_1 \neq$ Red
The Spaniard has a Dog → $A_1 \neq$ Dog
The Japanese is a Painter
The Italian drinks Tea
The Norwegian lives in the first house on the left → N_1 = Norwegian
The owner of the Green house drinks Coffee
The Green house is on the right of the White house
The Sculptor breeds Snails
The Diplomat lives in the Yellow house
The owner of the middle house drinks Milk → D_3 = Milk
The Norwegian lives next door to the Blue house
The Violinist drinks Fruit juice → $J_3 \neq$ Violinist
The Fox is in the house next to the Doctor's
The Horse is next to the Diplomat's

Task Scheduling



Four tasks T_1, T_2, T_3, and T_4 are related by time constraints:

- T_1 must be done during T_3
- T_2 must be achieved before T_1 starts
- T_2 must overlap with T_3
- T_4 must start after T_1 is complete

- Are the constraints compatible?
- What are the possible time relations between two tasks?
- What if the tasks use resources in limited supply?

How to formulate this problem as a CSP?

3-SAT

- n Boolean variables $u_1, \ldots, u_n$
- p constrains of the form

$$u_i^* \vee u_j^* \vee u_k^* = 1$$

where u* stands for either u or $\neg$u

- Known to be NP-complete

Types of variables in CSP formulation

- Discrete variables
  - Finite CSP: each variable has a finite domain of values
  - Infinite CSP: some or all variables have an infinite domain

    E.g., linear programming problems over the reals:

    $$\text{for } i = 1, 2, \ldots, p : a_{i,1}x_1 + a_{i,2}x_2 + \ldots + a_{i,n}x_n = a_{i,0}$$

    $$\text{for } j = 1, 2, \ldots, q : b_{j,1}x_1 + b_{j,2}x_2 + \ldots + b_{j,n}x_n \leq b_{j,0}$$

- Continuous variables
  - E.g., exact start/end times for Hubble Space Telescope observations

- We will only consider finite CSP

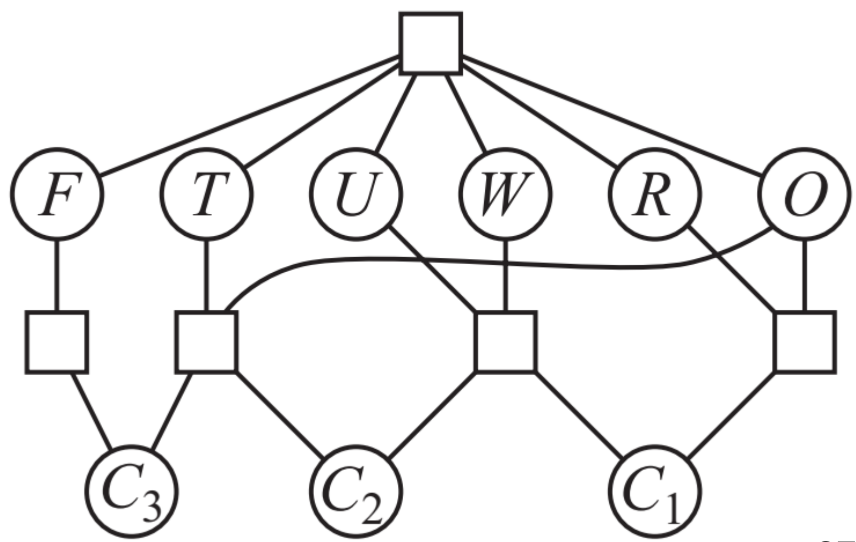
Types of constraints in CSP formulation

- Unary constraints involve a single variable
  - E.g., SA≠green
- Binary constraints involve pairs of variables
  - E.g., SA≠WA
- Higher-order constraints involve 3 or more variables
  - E.g., cryptarithmetic column constraints
  - Every higher-order finite constraint can be broken into n binary constraints, given enough auxiliary constraints
- Global constraint involves an arbitrary number of variables
  - E.g., Alldiff , which says that all of the variables involved in the constraint must have different values

Cryptarithmetic example

- **Variables:** $F, T, U, W, R, O, C_1, C_2, C_3$
- **Domains:** $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- **Constraints:** Alldiff (F,T,U,W,R,O)
  - $O + O = R + 10 \cdot C1$
  - $C1 + W + W = U + 10 \cdot C2$
  - $C2 + T + T = O + 10 \cdot C3$
  - $C3 = F, T \neq 0, F \neq 0$

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



Sudoku example

- **Variables:** Each (open) square
- **Domains:** {1, 2, 3, 4, 5, 6, 7, 8, 9}
- **Constraints:**
  - 9-way alldiff for each row
  - 9-way alldiff for each column
  - 9-way alldiff for each region

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

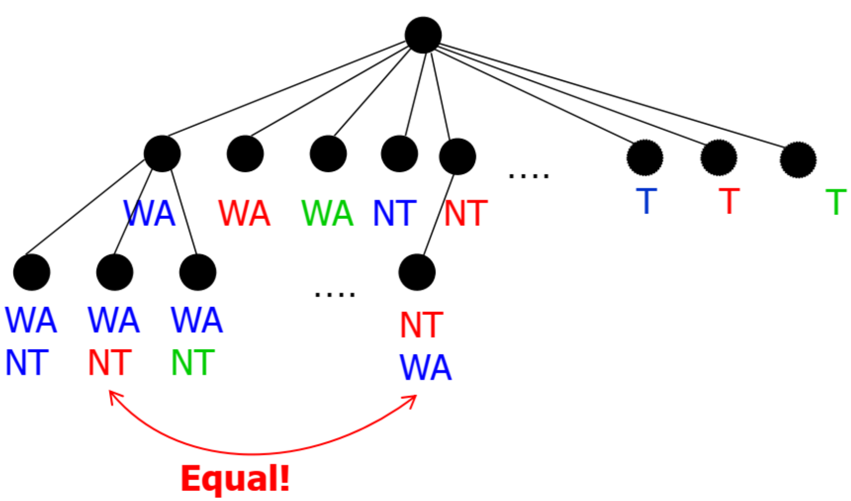
	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

Constraint Satisfaction Problems (CSPs)

- Standard search problem
  - State is a "black box" with no internal structure (it is a goal or not a goal)
- Solving CSPs more efficiently
  - State is specified by variables or features X_i (i=1, …, n) (factored representation)
  - Goal test: Whether each variable has a value that satisfies all the constraints on the variable?
- CSP search algorithms use general-purpose heuristics based on the structure of states

CSP as a Search Problem

- n variables $X_1, \ldots, X_n$
- Valid assignment: $\{X_{i1} \leftarrow v_{i1}, \ldots, X_{ik} \leftarrow v_{ik}\}$, $0 \leq k \leq n$, such that the values $v_{i1}, \ldots, v_{ik}$ satisfy all constraints relating the variables $X_{i1}, \ldots, X_{ik}$
- Complete assignment: one where k = n
  [if all variable domains have size d, there are $O(d^n)$ complete assignments]

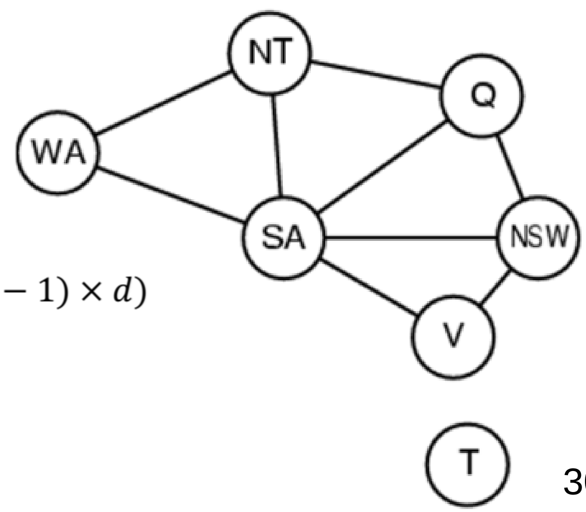


$n \times d$

$(n \times d) \times ((n-1) \times d)$

$\ldots$

$n! \times d^n$



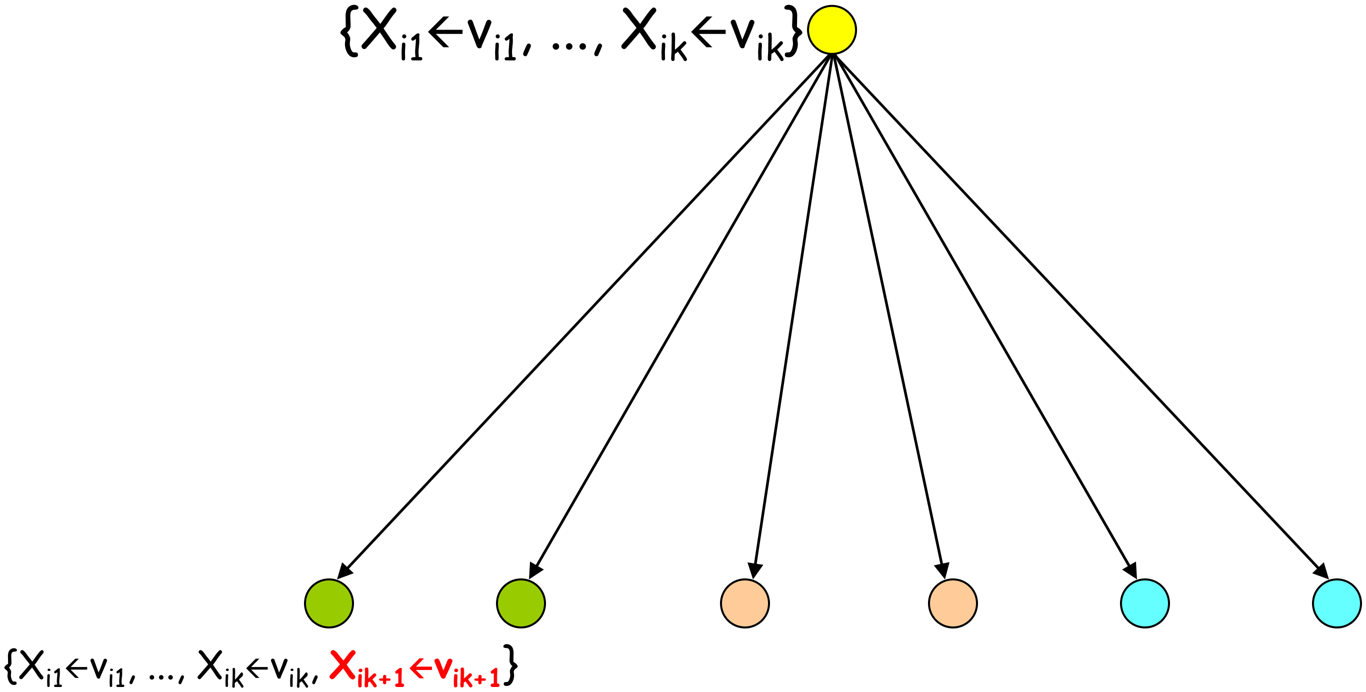
CSP as a Search Problem

- n variables $X_1, \ldots, X_n$
- Valid assignment: $\{X_{i1} \leftarrow v_{i1}, \ldots, X_{ik} \leftarrow v_{ik}\}$, $0 \leq k \leq n$, such that the values $v_{i1}, \ldots, v_{ik}$ satisfy all constraints relating the variables $X_{i1}, \ldots, X_{ik}$
- Complete assignment: one where $k = n$
  [if all variable domains have size d, there are $O(d^n)$ complete assignments]
- States: valid assignments

CSP as a Search Problem

- n variables $X_1, \ldots, X_n$
- Valid assignment: $\{X_{i1} \leftarrow v_{i1}, \ldots, X_{ik} \leftarrow v_{ik}\}$, $0 \leq k \leq n$, such that the values $v_{i1}, \ldots, v_{ik}$ satisfy all constraints relating the variables $X_{i1}, \ldots, X_{ik}$
- Complete assignment: one where k = n
  [if all variable domains have size d, there are $O(d^n)$ complete assignments]
- States: valid assignments
- Initial state: empty assignment {}, i.e. k = 0
- Successor of a state:

  $\{X_{i1} \leftarrow v_{i1}, \ldots, X_{ik} \leftarrow v_{ik}\} \rightarrow \{X_{i1} \leftarrow v_{i1}, \ldots, X_{ik} \leftarrow v_{ik}, \mathbf{X_{ik+1} \leftarrow v_{ik+1}}\}$
- Goal test: k = n



$r = n-k$ variables with s values → $r \times s$ branching factor

A Key property of CSP: Commutativity

The order in which variables are assigned values has no impact on the reachable complete valid assignments

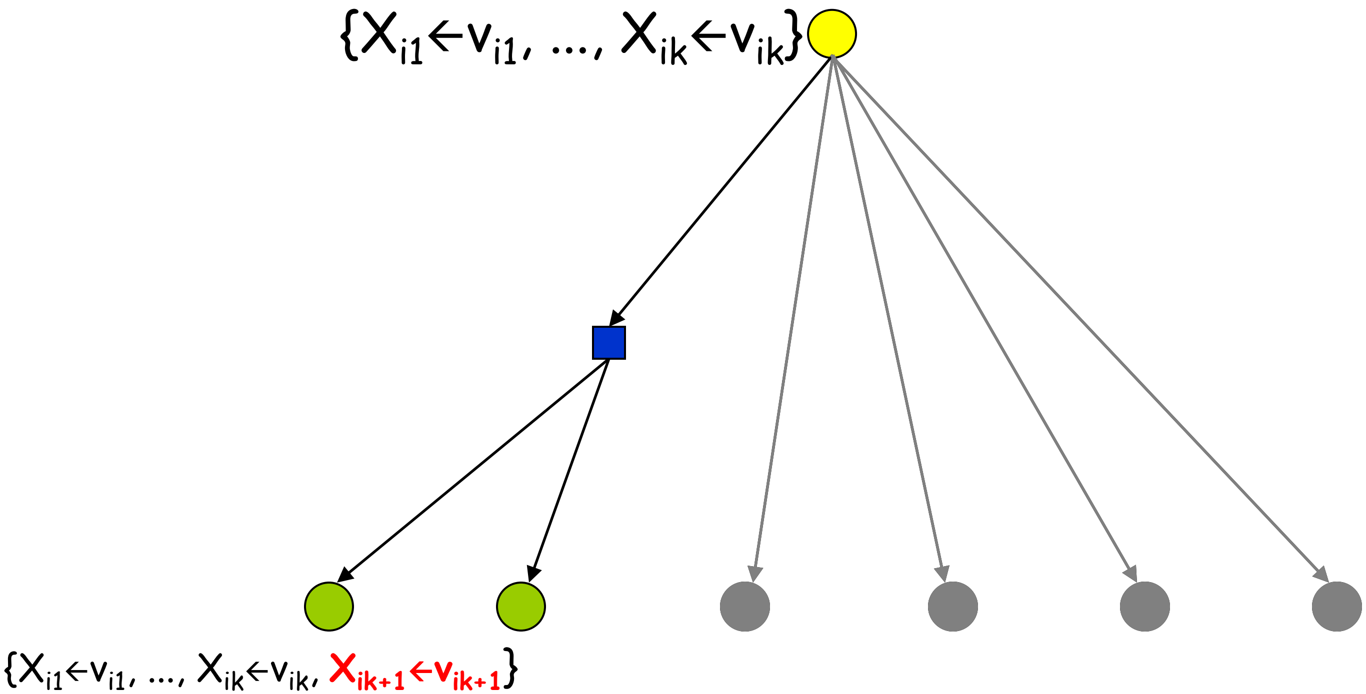
Hence:

1) One can expand a node N by first selecting **one** variable X not in the assignment *A* associated with N and then assigning every value v in the domain of X
[→ big reduction in branching factor]

- 4 variables $X_1, \ldots, X_4$
- Let the valid assignment of N be:

  $$A = \{X_1 \leftarrow v_1, X_3 \leftarrow v_3\}$$
- For example pick variable X_4
- Let the domain of X_4 be $\{v_{4,1}, v_{4,2}, v_{4,3}\}$
- The successors of A are all the valid assignments among:

  $$\{X_1 \leftarrow v_1, X_3 \leftarrow v_3 , X_4 \leftarrow v_{4,1}\}$$
  $$\{X_1 \leftarrow v_1, X_3 \leftarrow v_3 , X_4 \leftarrow v_{4,2}\}$$
  $$\{X_1 \leftarrow v_1, X_3 \leftarrow v_3 , X_4 \leftarrow v_{4,2}\}$$



$r = n - k$ variables with s values $\rightarrow$ **s** branching factor

The depth of the solutions in the search tree is un-changed (n)

A Key property of CSP: Commutativity

The order in which variables are assigned values has no impact on the reachable complete valid assignments

Hence:

1) One can expand a node N by first selecting **one** variable X not in the assignment *A* associated with N and then assigning every value v in the domain of X
   [→ big reduction in branching factor]

2) One need not store the path to a node
   → Backtracking search algorithm

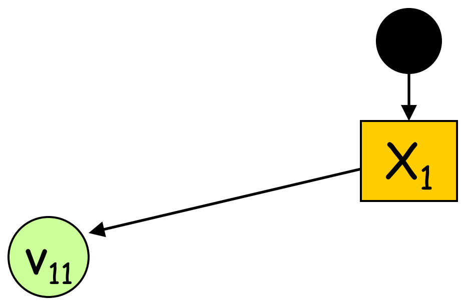
Backtracking Search

Essentially a simplified depth-first algorithm using recursion

Backtracking Search
(3 variables)

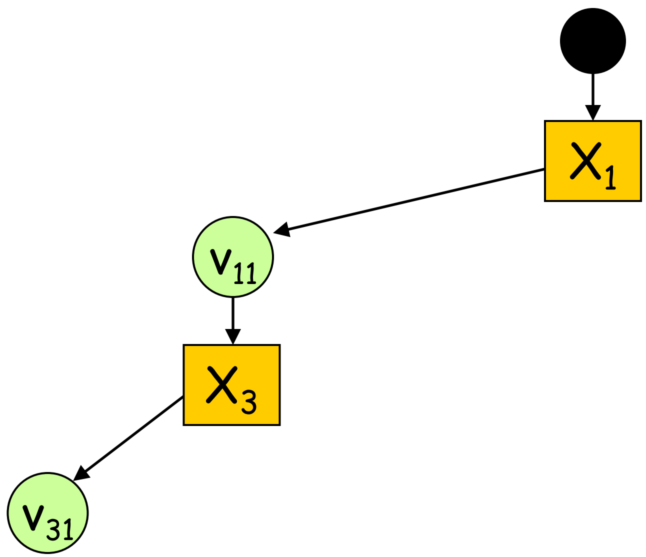
Assignment = {}

Backtracking Search
(3 variables)



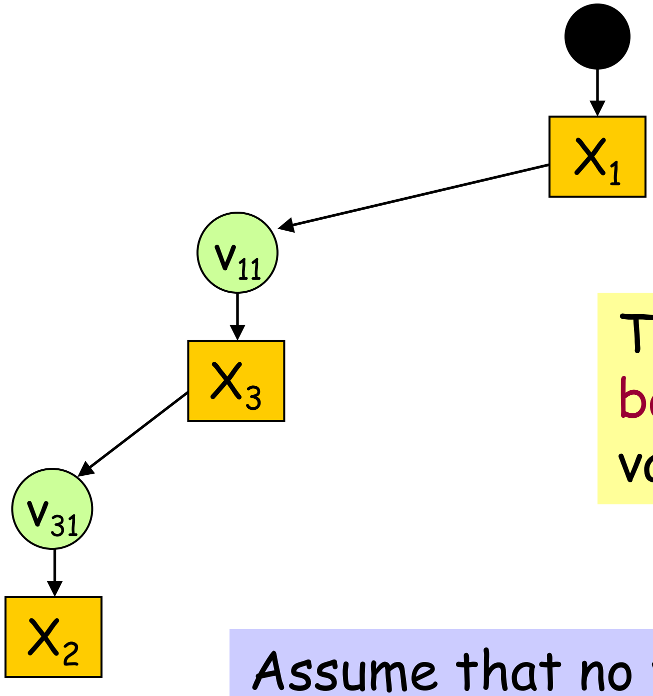
Assignment = $\{(X_1, v_{11})\}$

Backtracking Search
(3 variables)



Assignment = $\{(X_1, v_{11}), (X_3, v_{31})\}$

Backtracking Search
(3 variables)

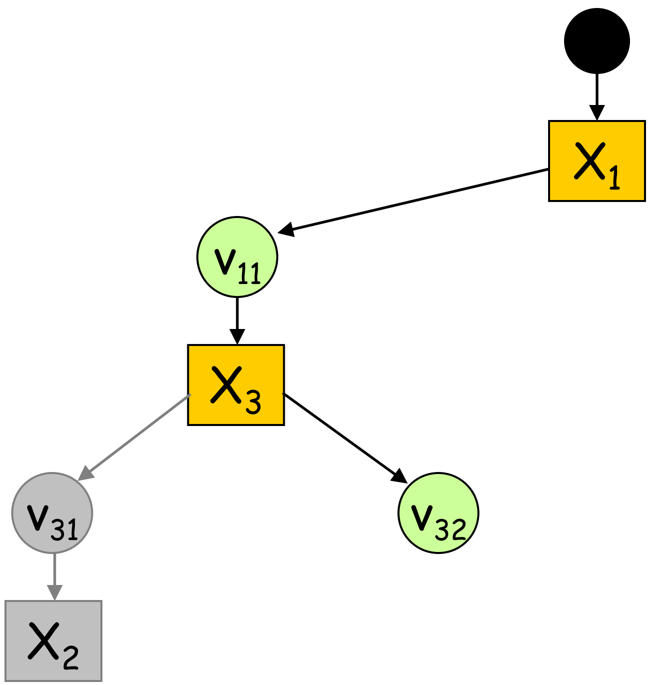


Then, the search algorithm backtracks to the previous (X_3) variable and tries another value

Assume that no value of X_2 leads to a valid assignment

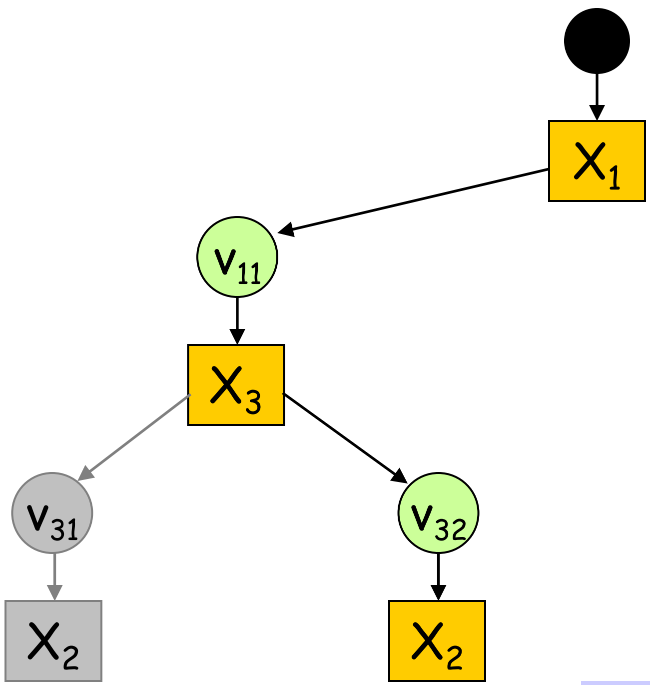
Assignment = $\{(X_1,v_{11}), (X_3,v_{31})\}$

Backtracking Search
(3 variables)



Assignment = $\{(X_1, v_{11}), (X_3, v_{32})\}$

Backtracking Search
(3 variables)

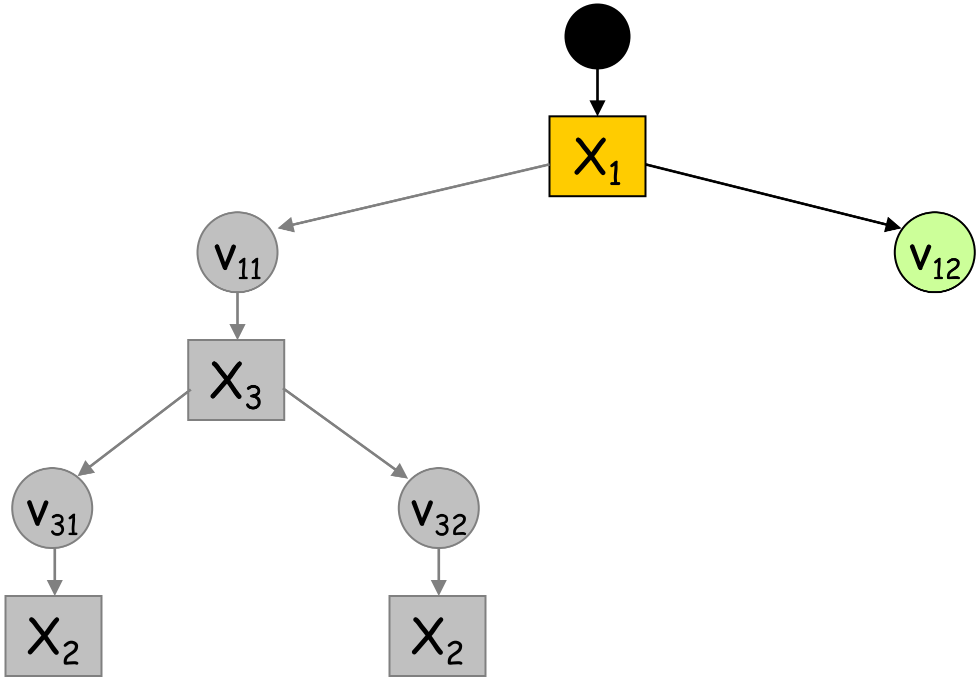


The search algorithm backtracks to the previous variable (X_3) and tries another value. But assume that X_3 has only two possible values. The algorithm backtracks to X_1

Assume again that no value of X_2 leads to a valid assignment

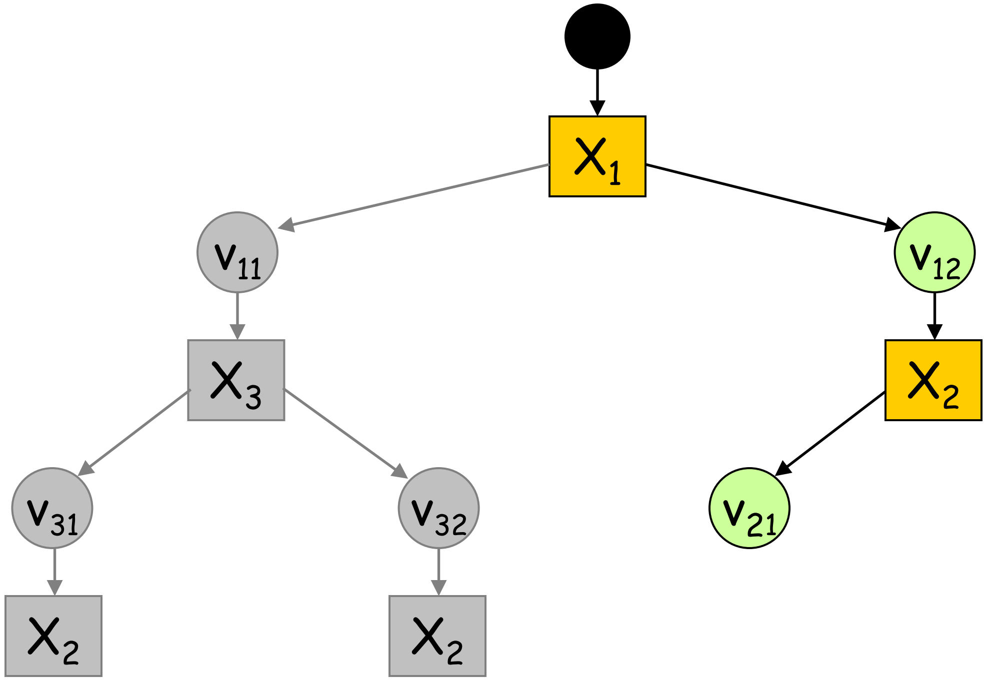
Assignment = $\{(X_1, v_{11}), (X_3, v_{32})\}$

Backtracking Search
(3 variables)



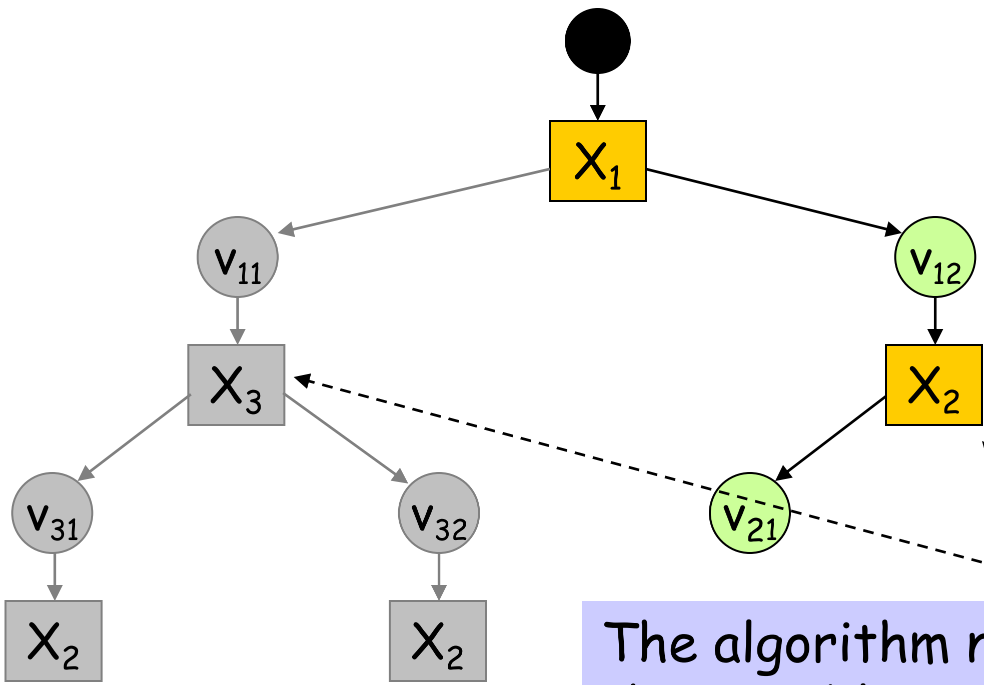
Assignment = $\{(X_1, v_{12})\}$

Backtracking Search
(3 variables)



Assignment = $\{(X_1,v_{12}), (X_2,v_{21})\}$

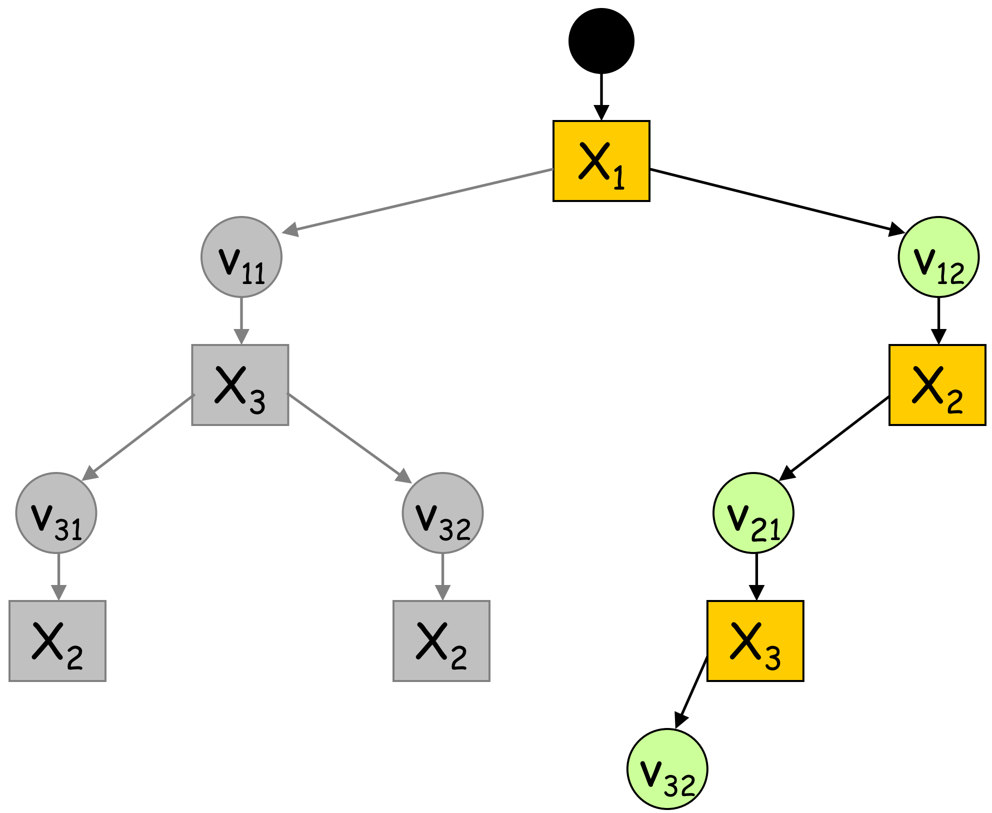
Backtracking Search
(3 variables)



The algorithm need not consider the variables in the same order in this sub-tree as in the other

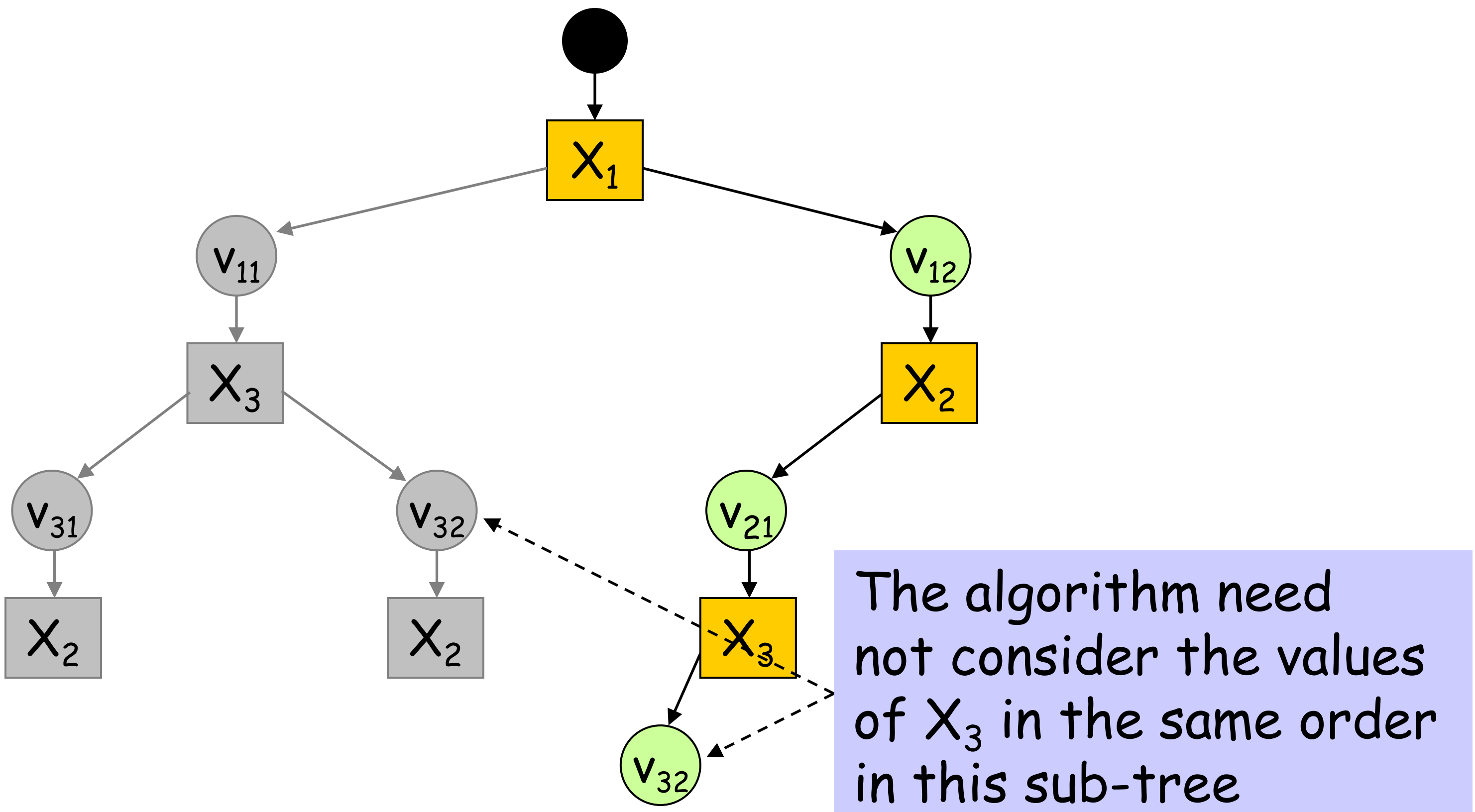
Assignment = $\{(X_1,v_{12}), (X_2,v_{21})\}$

Backtracking Search
(3 variables)



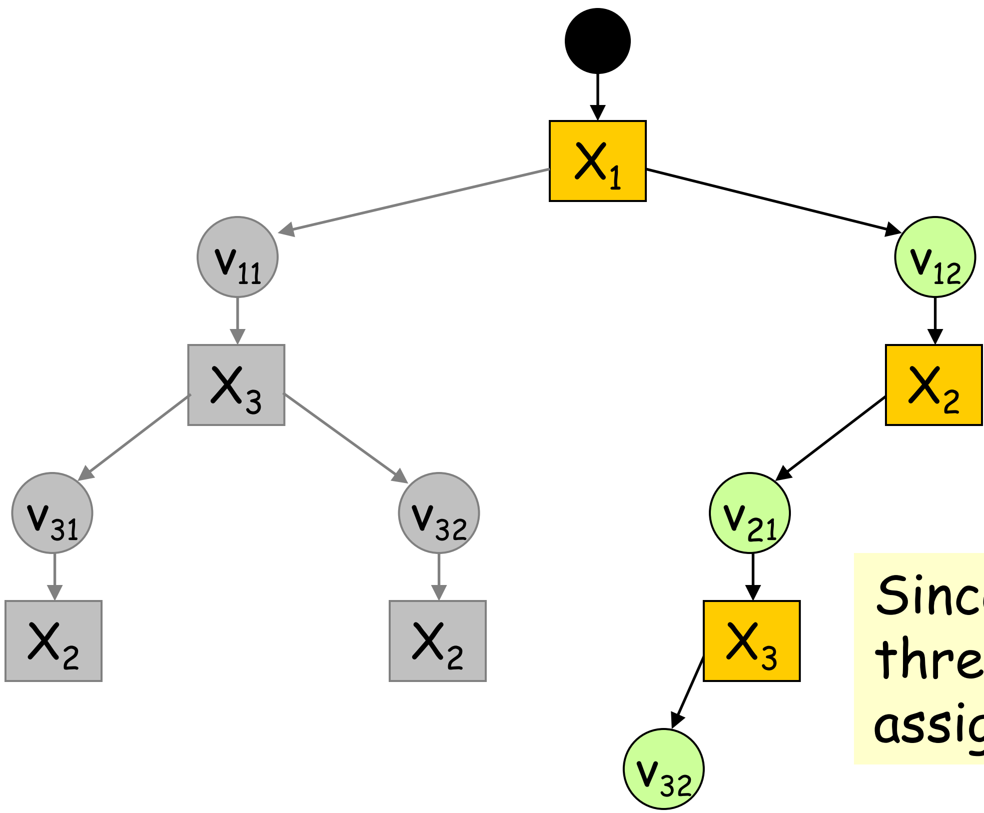
Assignment = $\{(X_1,v_{12}), (X_2,v_{21}), (X_3,v_{32})\}$

Backtracking Search
(3 variables)



Assignment = $\{(X_1,v_{12}), (X_2,v_{21}), (X_3,v_{32})\}$

Backtracking Search
(3 variables)



Since there are only three variables, the assignment is complete

Assignment = $\{(X_1,v_{12}), (X_2,v_{21}), (X_3,v_{32})\}$

Backtracking Algorithm

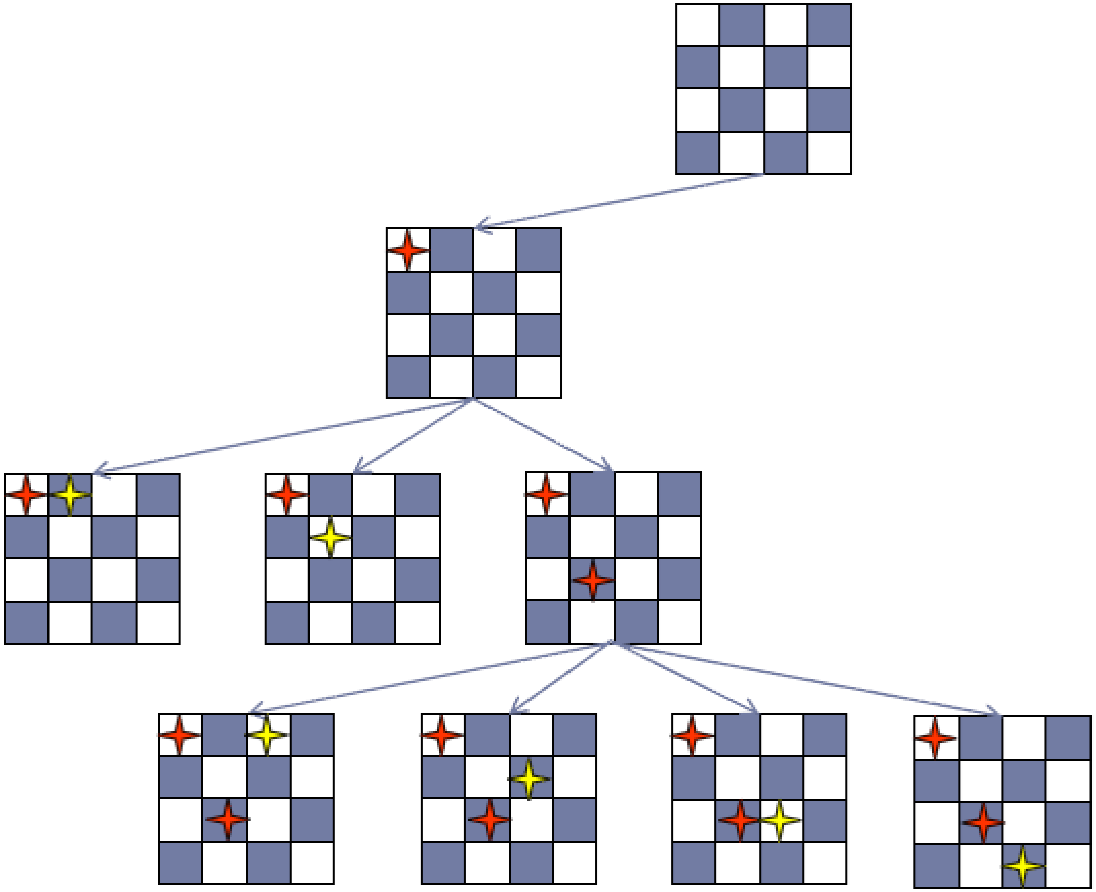
CSP-BACKTRACKING(A)

1. If assignment A is complete then return A
2. X ← select a variable not in A
3. D ← select an ordering on the domain of X
4. For each value v in D do
   a. Add (X←v) to A
   b. If A is valid then
      i. result ← CSP-BACKTRACKING(A)
      ii. If result ≠ failure then return result
   c. Remove (X←v) from A
5. Return failure

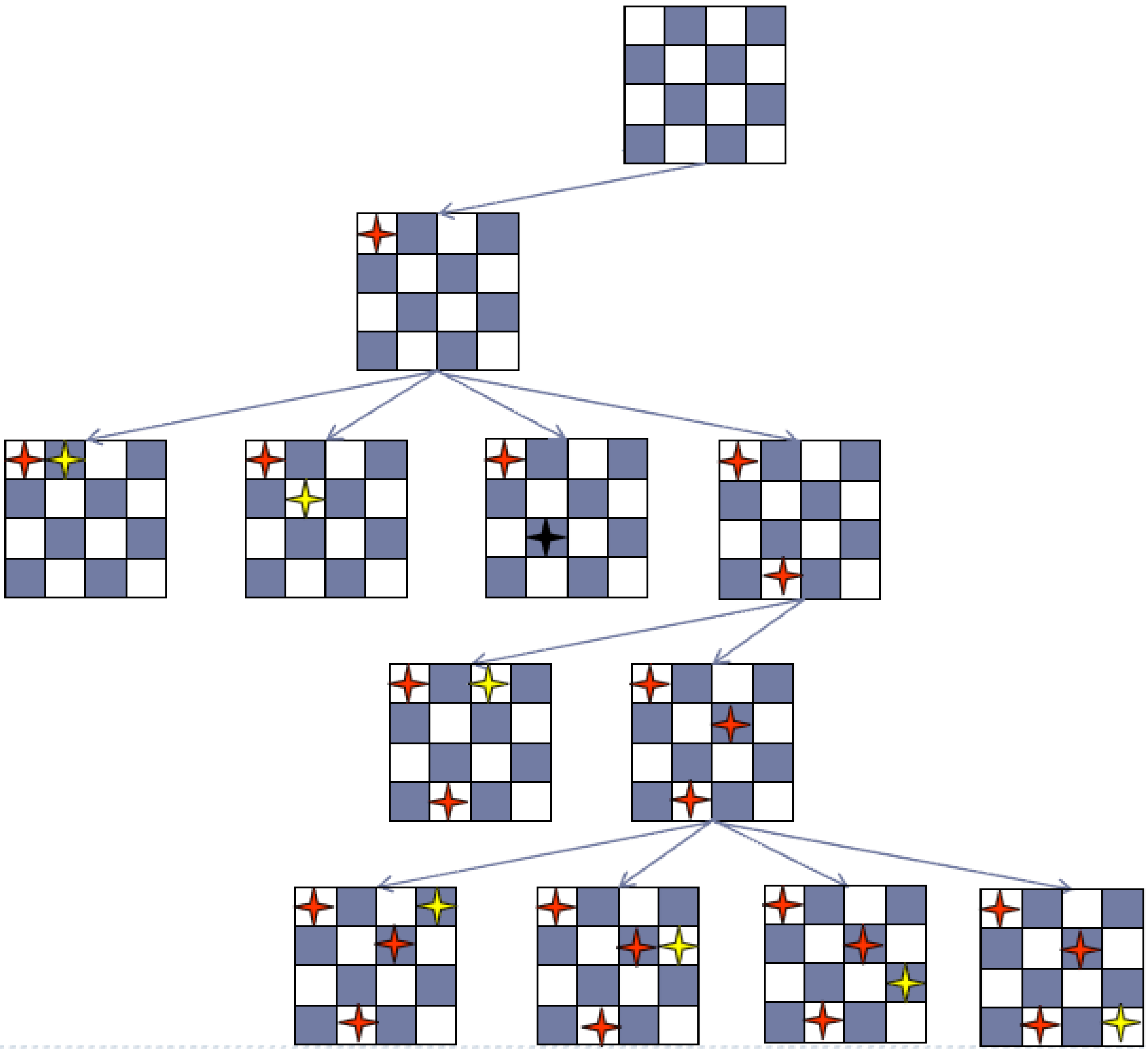
Call CSP-BACKTRACKING({})

[This recursive algorithm keeps too much data in memory.
An iterative version could save memory (left as an exercise)]

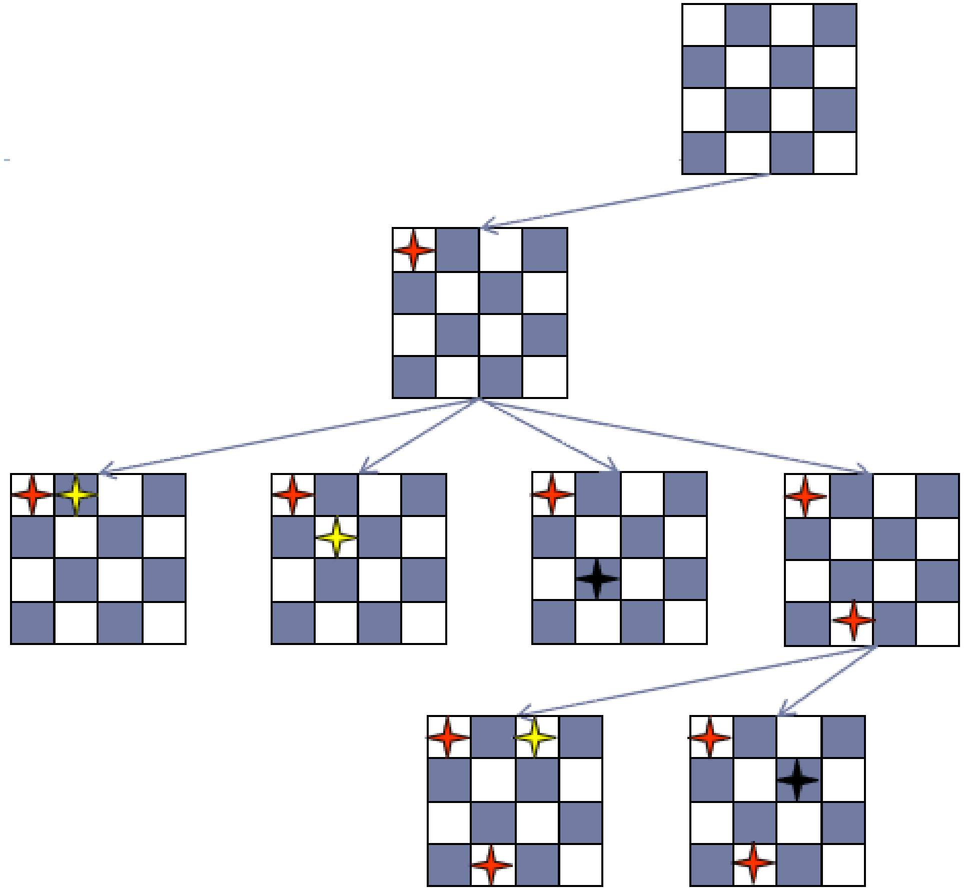
4-Queens example



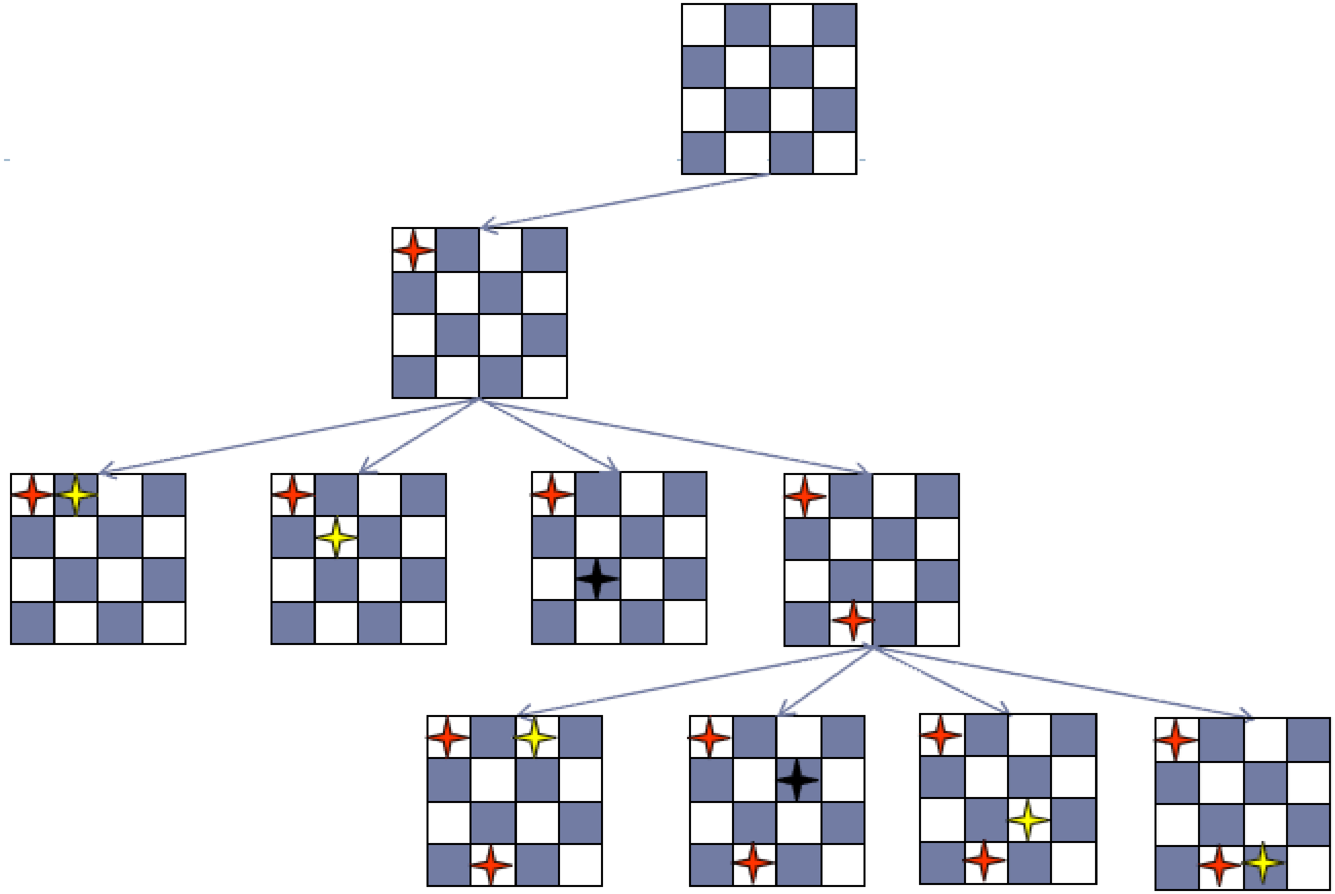
4-Queens example



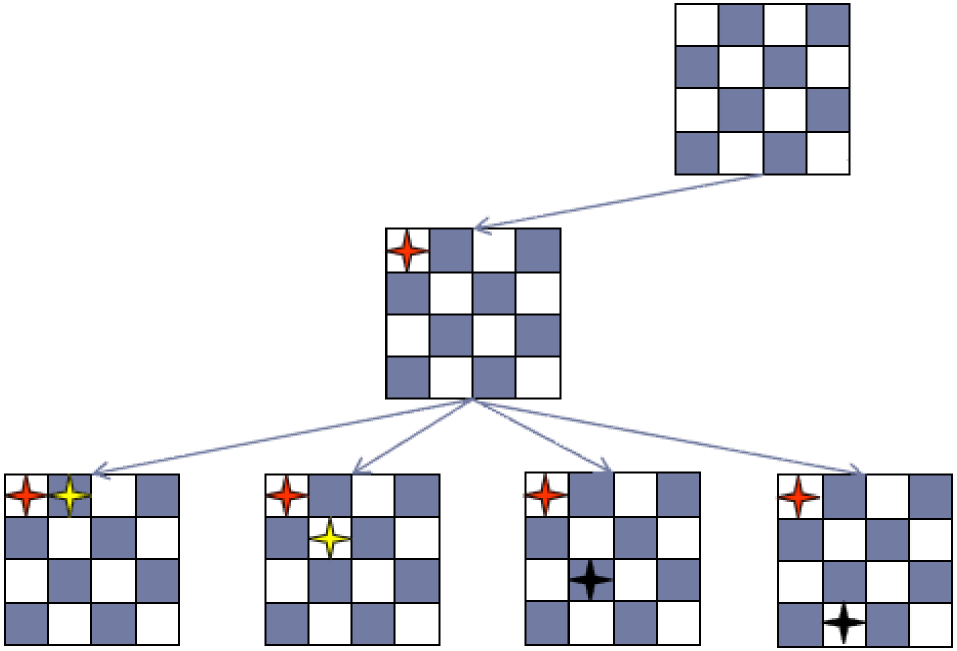
4-Queens example



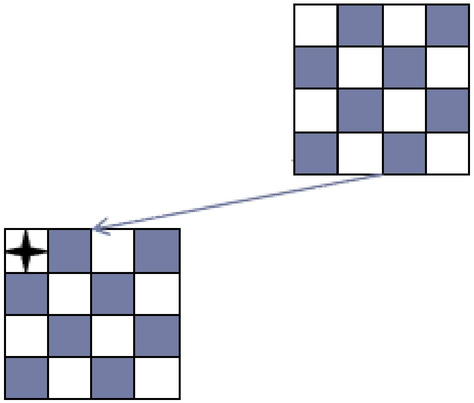
4-Queens example



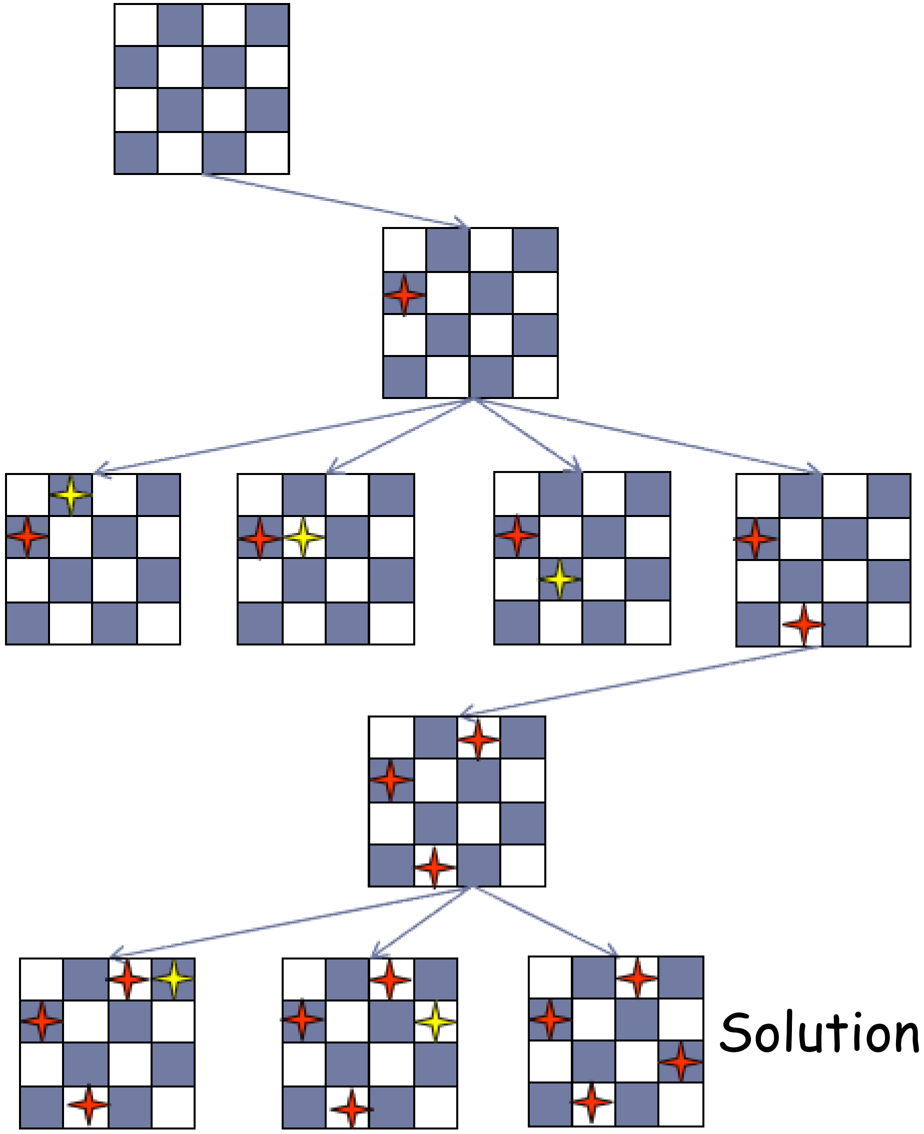
4-Queens example



4-Queens example

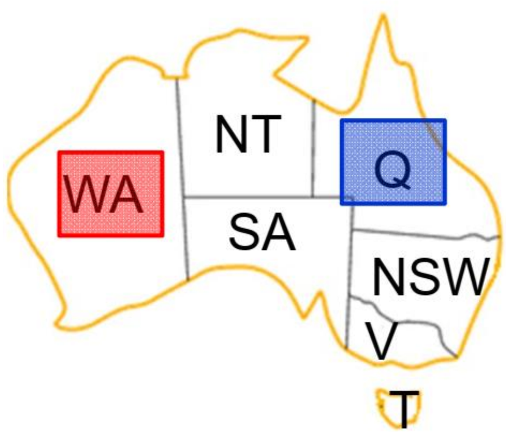


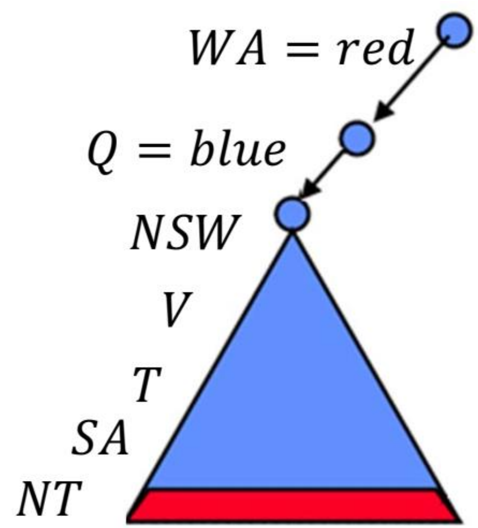
4-Queens example



Naïve backtracking (late failure)

- Map coloring with three colors
  - {WA=red, Q=blue} can not be completed.
  - However, the backtracking search does not detect this before selecting NT and SA variables





Critical Questions for the Efficiency of CSP-Backtracking

CSP-BACKTRACKING(A)

1. If assignment A is complete then return A
2. X ← **select** a variable not in A
3. D ← **select** an ordering on the domain of X
4. For each value v in D do
    a. Add (X←v) to A
    b. If a is valid then
        i. result ← CSP-BACKTRACKING(A)
        ii. If result ≠ failure then return result
    c. Remove (X←v) from A
5. Return failure

Critical Questions for the Efficiency of CSP-Backtracking

1) Which variable X should be assigned a value next?

2) In which order should X's values be assigned?

Critical Questions for the Efficiency of CSP-Backtracking

1) Which variable X should be assigned a value next?
   The current assignment may not lead to any solution, but the algorithm does not know it yet. Selecting the right variable X may help discover the contradiction more quickly

2) In which order should X's values be assigned?

Critical Questions for the Efficiency of CSP-Backtracking

1) Which variable X should be assigned a value next?
   The current assignment may not lead to any solution, but the algorithm does not know it yet. Selecting the right variable X may help discover the contradiction more quickly

2) In which order should X's values be assigned?
   The current assignment may be part of a solution. Selecting the right value to assign to X may help discover this solution more quickly

Critical Questions for the Efficiency of CSP-Backtracking

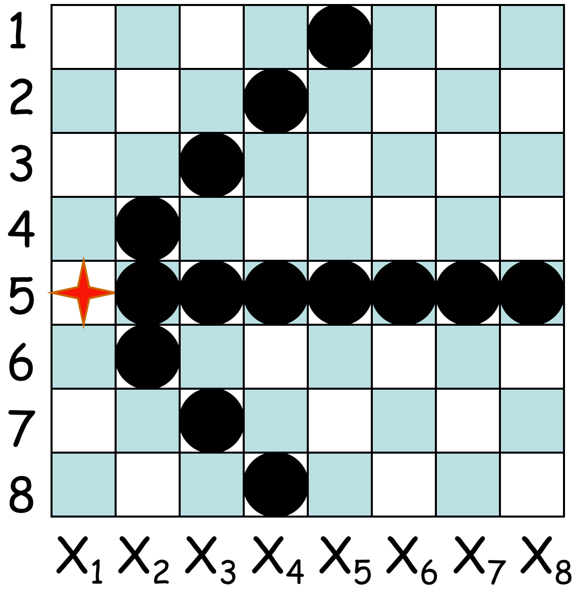
1) Which variable X should be assigned a value next?
   The current assignment may not lead to any solution, but the algorithm does not know it yet. Selecting the right variable X may help discover the contradiction more quickly

2) In which order should X's values be assigned?
   The current assignment may be part of a solution. Selecting the right value to assign to X may help discover this solution more quickly

More on these questions very soon ...

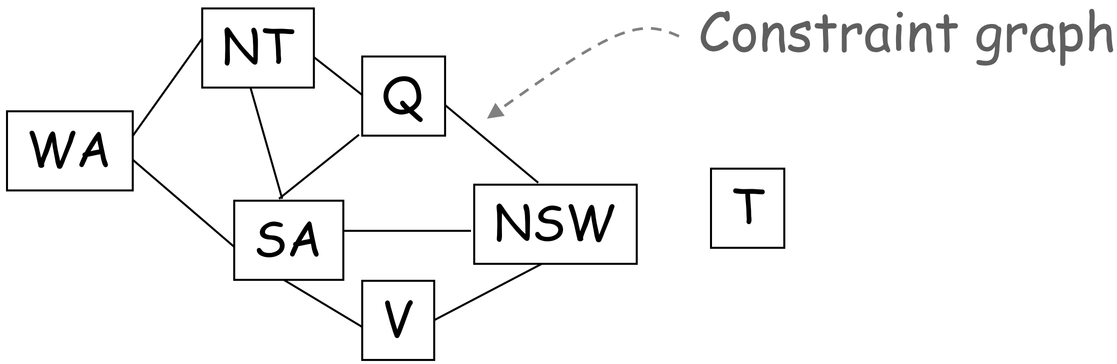
Forward Checking

A simple constraint-propagation technique:



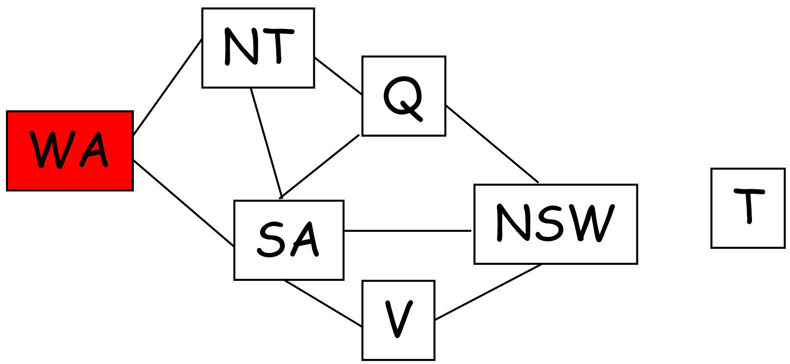
Assigning the value 5 to X_1 leads to removing values from the domains of X_2, X_3, ..., X_8

Forward Checking in Map Coloring



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

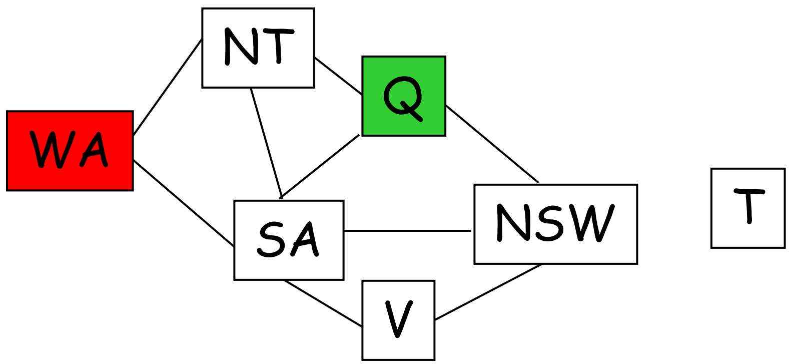
Forward Checking in Map Coloring



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

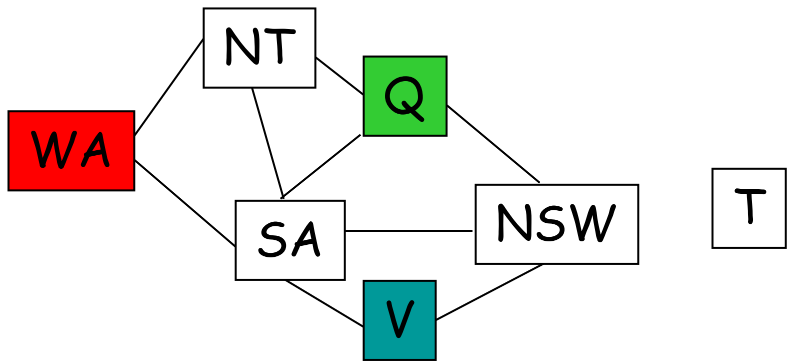
Forward checking removes the value Red of NT and of SA

Forward Checking in Map Coloring



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

Forward Checking in Map Coloring



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

Forward Checking in Map Coloring

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

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

Forward Checking (General Form)

Whenever a pair ($X \leftarrow v$) is added to assignment *A* do:

For each variable Y not in *A* do:

For every constraint *C* relating Y to the variables in *A* do:

Remove all values from Y's domain that do not satisfy *C*

Modified Backtracking Algorithm

CSP-BACKTRACKING(A, var-domains)

1. If assignment A is complete then return A
2. X ← select a variable not in A
3. D ← select an ordering on the domain of X
4. For each value v in D do
   a. Add (X←v) to A
   b. var-domains ← forward checking(var-domains, X, v, A)
   c. If a variable has an empty domain then return failure
   d. result ← CSP-BACKTRACKING(A, var-domains)
   e. If result ≠ failure then return result
   f. Remove (X←v) from A
5. Return failure

Modified Backtracking Algorithm

CSP-BACKTRACKING(A, var-domains)

1. If assignment A is complete then return A
2. X ← select a variable not in A
3. D ← select an ordering on the domain of X
4. For each value v in D do
   a. Add (X←v) to A
   b. var-domains ← forward checking(var-domains, X, v, A)
   c. If a variable has an empty domain then return failure
   d. result ← CSP-BACKTRACKING(A, var-domains)
   e. If result ≠ failure then return result
   f. Remove (X←v) from A
5. Return failure

No need any more to verify that A is valid

Modified Backtracking Algorithm

CSP-BACKTRACKING(A, var-domains)

1. If assignment A is complete then return A
2. X ← select a variable not in A
3. D ← select an ordering on the domain of X
4. For each value v in D do
   a. Add (X←v) to A
   b. var-domains ← forward checking(var-domains, X, v, A)
   c. If a variable has an empty domain then return failure
   d. result ← CSP-BACKTRACKING(A, var-domains)
   e. If result ≠ failure then return result
   f. Remove (X←v) from A
5. Return failure

Need to pass down the updated variable domains

Modified Backtracking Algorithm

CSP-BACKTRACKING(A, var-domains)

1. If assignment A is complete then return A
2. X ← select a variable not in A
3. D ← select an ordering on the domain of X
4. For each value v in D do
   a. Add (X←v) to A
   b. var-domains ← forward checking(var-domains, X, v, A)
   c. If a variable has an empty domain then return failure
   d. result ← CSP-BACKTRACKING(A, var-domains)
   e. If result ≠ failure then return result
   f. Remove (X←v) from A
5. Return failure

1) Which variable X_i should be assigned a value next?
→ Most-constrained-variable heuristic (MRV)
→ Most-constraining-variable heuristic

2) In which order should its values be assigned?
→ Least-constraining-value heuristic

These heuristics can be quite confusing

Keep in mind that **all** variables must eventually get a value, while only **one** value from a domain must be assigned to each variable

Most-Constrained-Variable (MCV) Heuristic

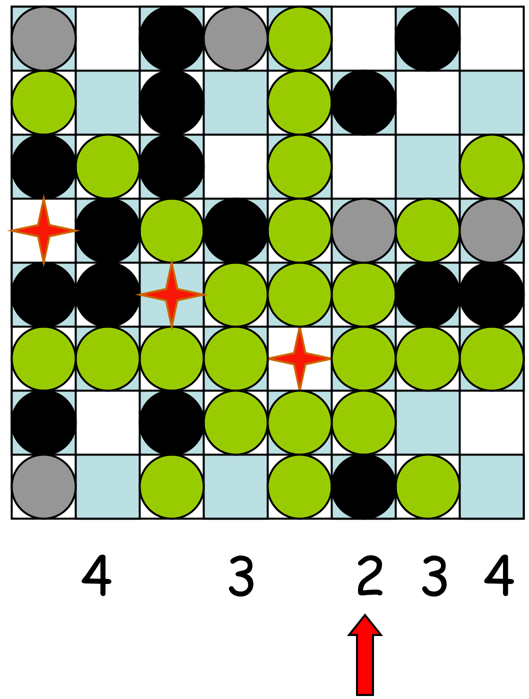
1) Which variable X_i should be assigned a value next?

Select the variable with the smallest remaining domain

[Rationale: Minimize the branching factor]

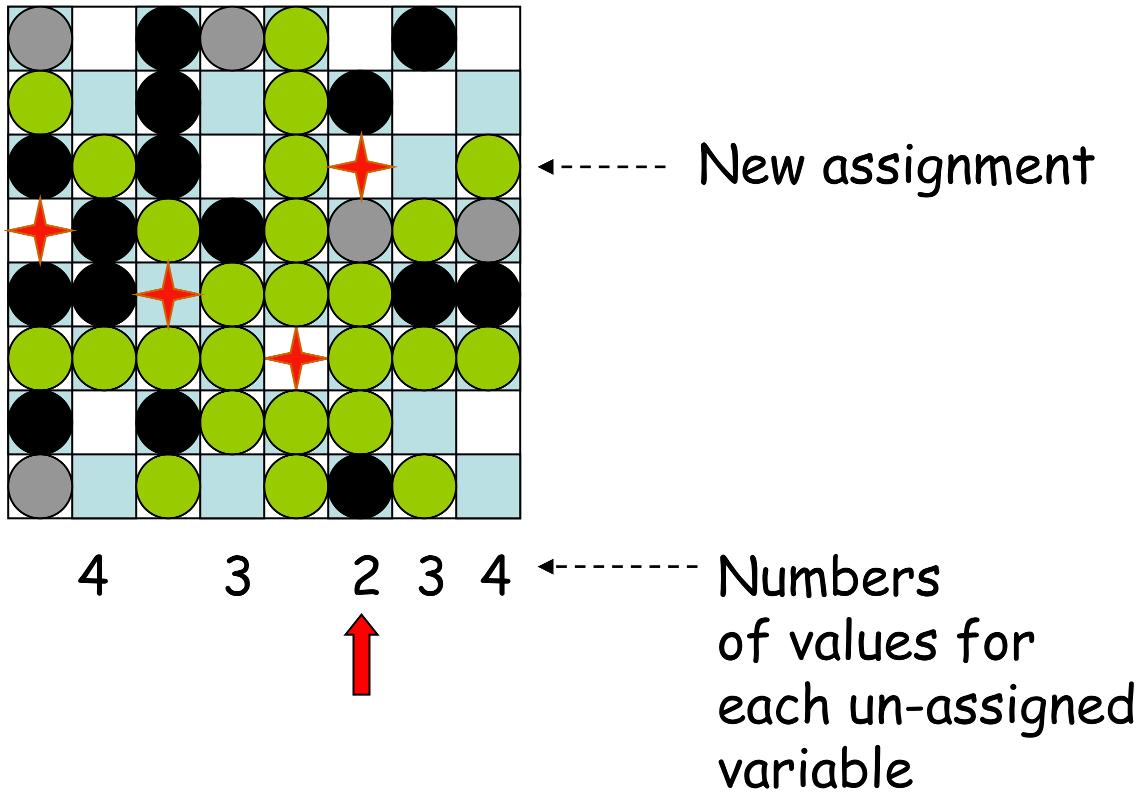
Also known as Minimum Remaining Value (MRV)

8-Queens

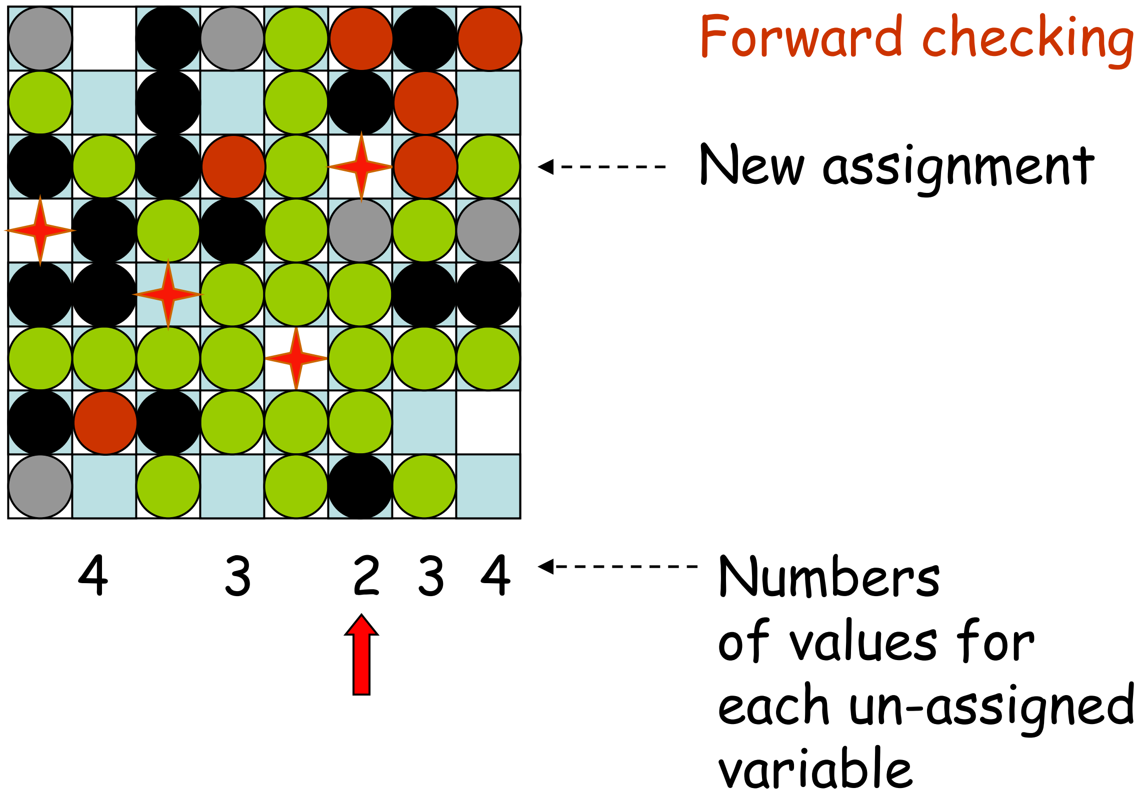


Numbers of values for each un-assigned variable

8-Queens

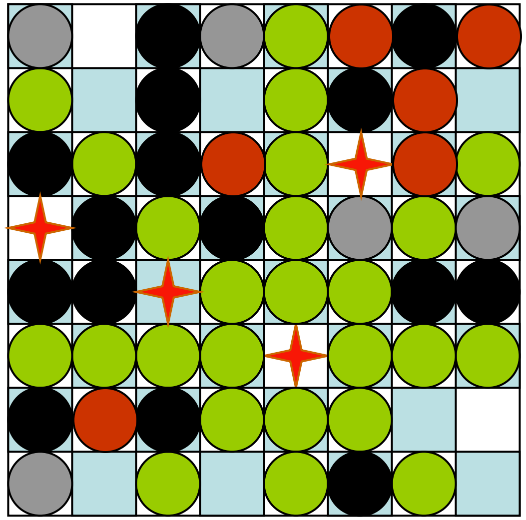

New assignment
4 3 2 3 4
Numbers of values for each un-assigned variable

8-Queens



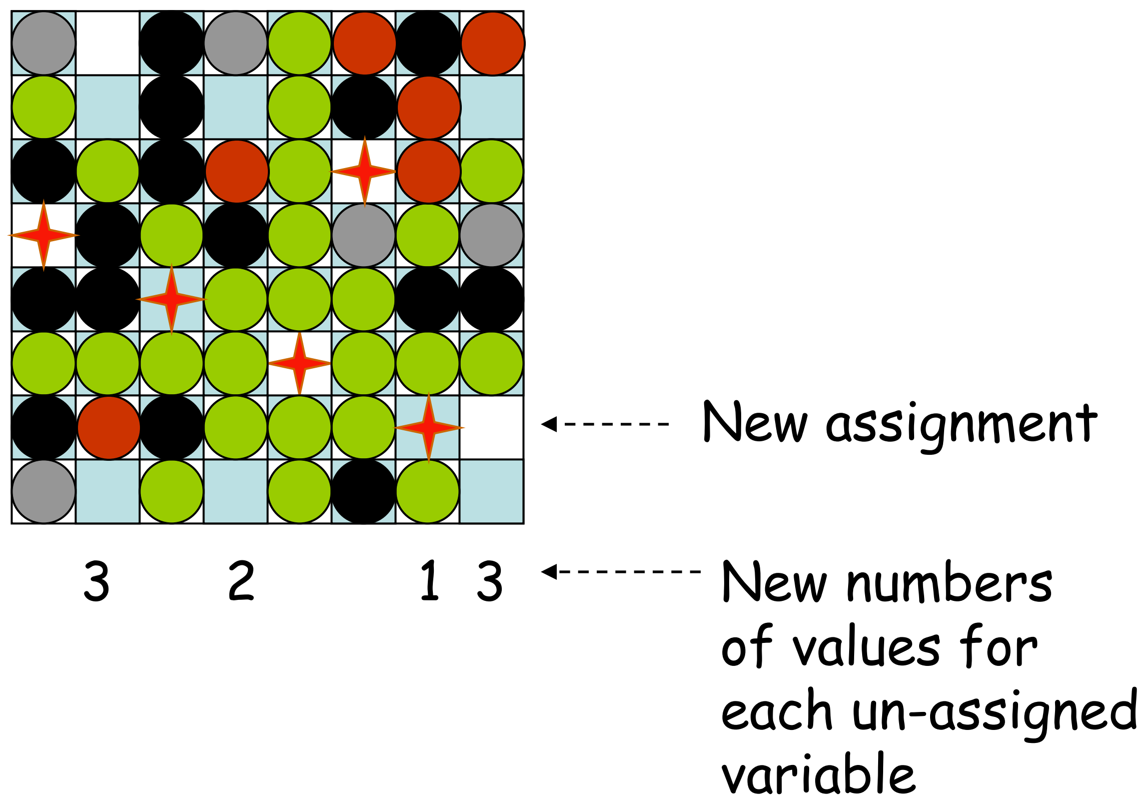
Forward checking
New assignment
4 3 2 3 4
Numbers of values for each un-assigned variable

8-Queens



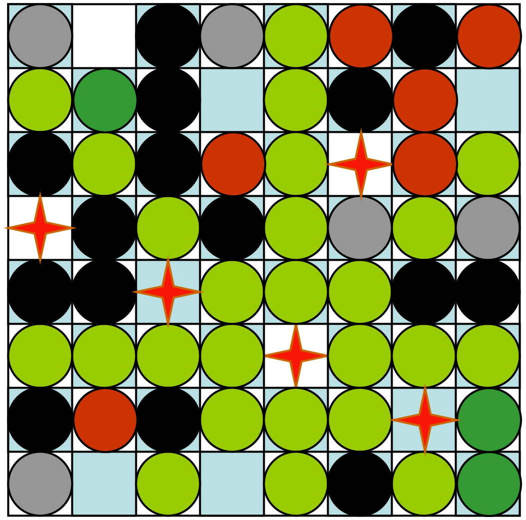
3 2 1 3 ← New numbers of values for each un-assigned variable

8-Queens



New assignment
3 2 1 3
New numbers of values for each un-assigned variable

8-Queens



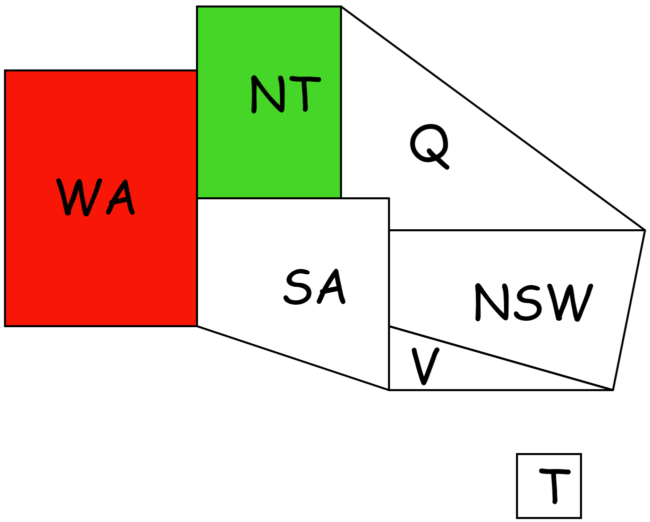
3 2 1 3

Forward checking

New assignment

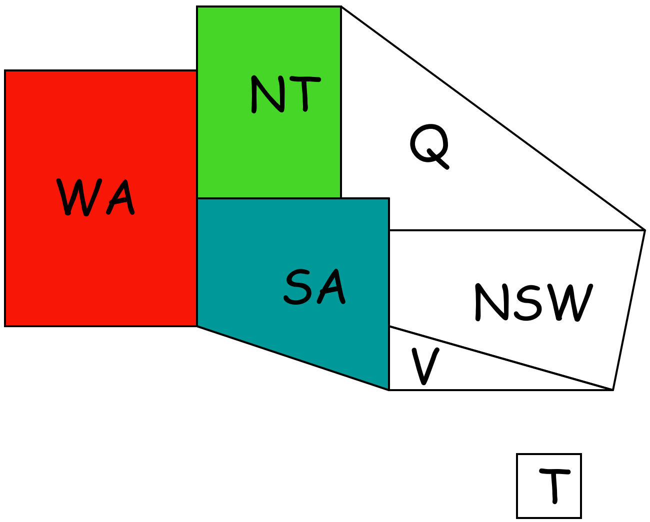
New numbers of values for each un-assigned variable

Map Coloring



- SA's remaining domain has size 1 (value Blue remaining)
- Q's remaining domain has size 2
- NSW's, V's, and T's remaining domains have size 3

Map Coloring



- SA's remaining domain has size 1 (value Blue remaining)
- Q's remaining domain has size 2
- NSW's, V's, and T's remaining domains have size 3

→ Select SA

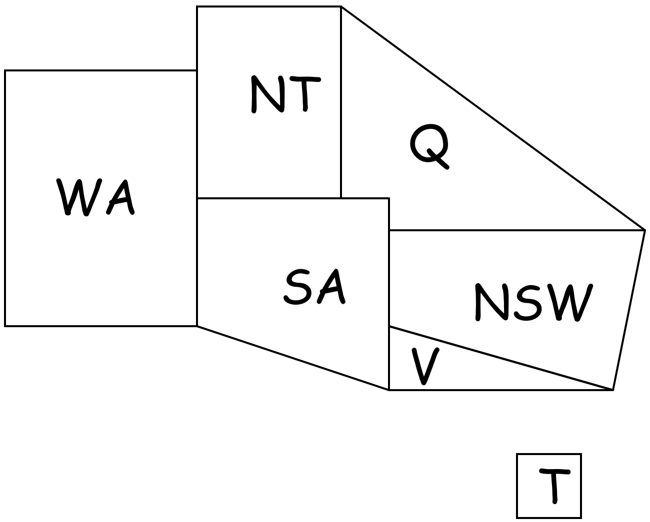
Most-Constraining-Variable Heuristic

1) Which variable X_i should be assigned a value next?

Among the variables with the smallest remaining domains (ties with respect to the most-constrained-variable heuristic), select the one that appears in the largest number of constraints on variables not in the current assignment (Degree heuristic)

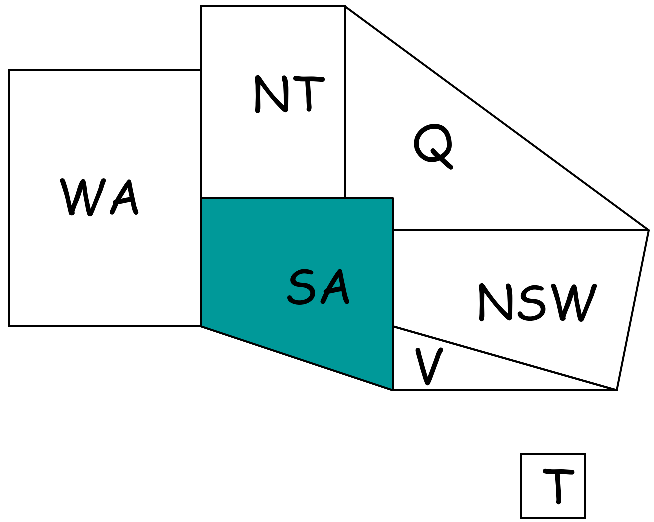
[Rationale: Increase future elimination of values, to reduce future branching factors]

Map Coloring



- Before any value has been assigned, all variables have a domain of size 3, but SA is involved in more constraints (5) than any other variable

Map Coloring



- Before any value has been assigned, all variables have a domain of size 3, but SA is involved in more constraints (5) than any other variable

→ Select SA and assign a value to it (e.g., Blue)

Least-Constraining-Value (LCV) Heuristic

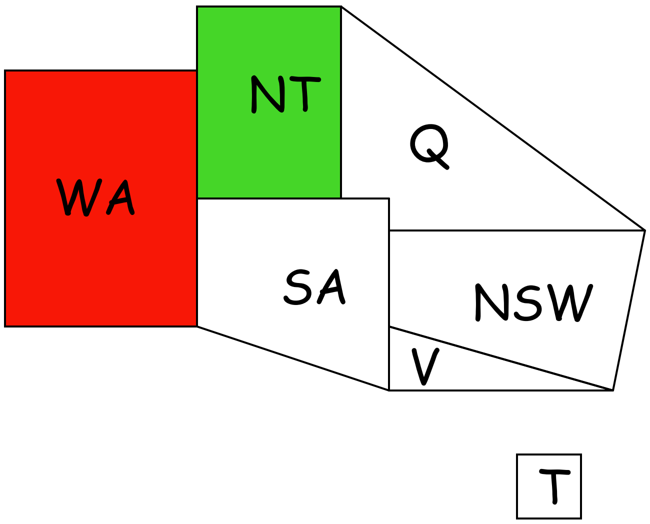
2) In which order should X's values be assigned?

Select the value of X that removes the smallest number of values from the domains of those variables which are not in the current assignment

[Rationale: Since only one value will eventually be assigned to X, pick the least-constraining value first, since it is the most likely not to lead to an invalid assignment]

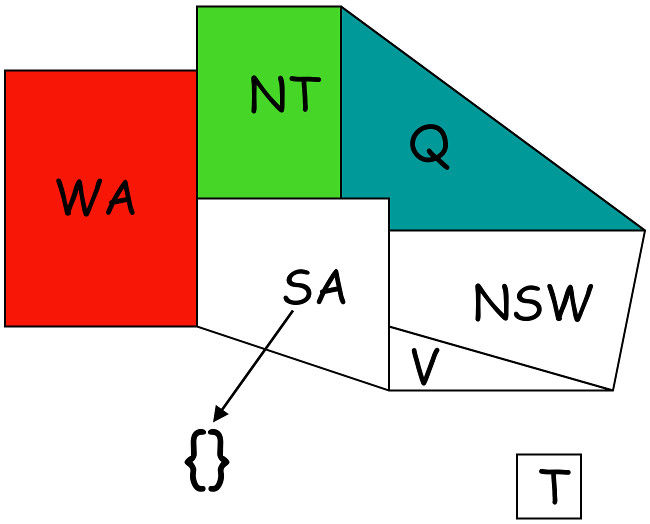
[Note: Using this heuristic requires performing a forward-checking step for every value, not just for the selected value]

Map Coloring



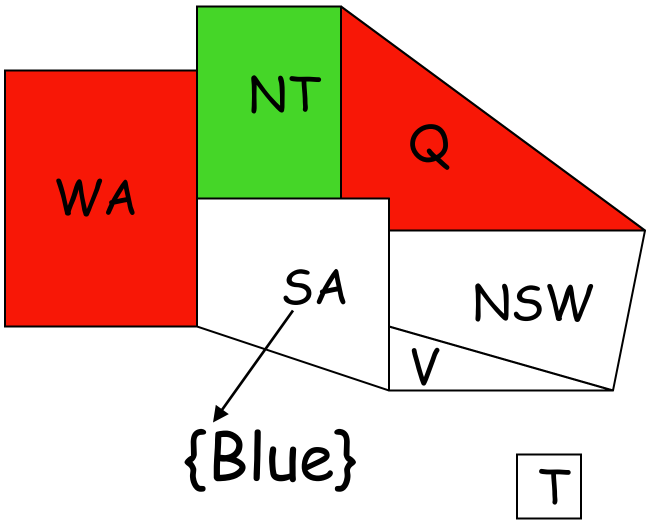
- Q's domain has two remaining values: Blue and Red
- Assigning Blue to Q would leave 0 value for SA, while assigning Red would leave 1 value

Map Coloring



- Q's domain has two remaining values: Blue and Red
- Assigning Blue to Q would leave 0 value for SA, while assigning Red would leave 1 value

Map Coloring



- Q's domain has two remaining values: Blue and Red
- Assigning Blue to Q would leave 0 value for SA, while assigning Red would leave 1 value

→ So, assign Red to Q

Modified Backtracking Algorithm

CSP-BACKTRACKING(A, var-domains)

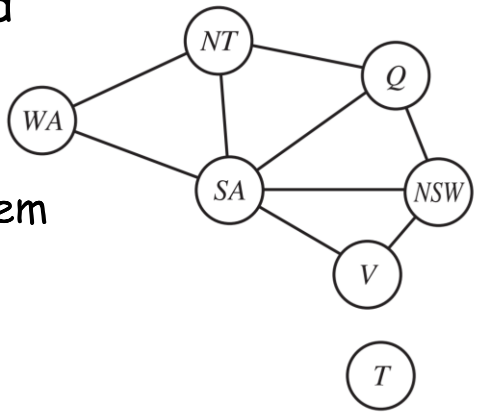
1. If assignment A is complete then return A
2. X ← select a variable not in A
3. D ← select an ordering on the domain of X
4. For each value v in D do
   a. Add (X←v) to A
   b. var-domains ← forward checking(var-domains, X, v, A)
   c. If a variable has an empty domain then return failure
   d. result ← CSP-BACKTRACKING(A, var-domains)
   e. If result ≠ failure then return result
   f. Remove (X←v) from A
5. Return failure

1) Most-constrained-variable heuristic
2) Most-constraining-variable heuristic

3) Least-constraining-value heuristic

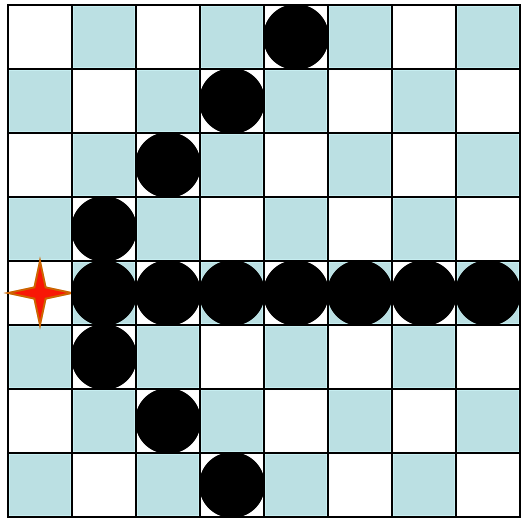
CSPs solver phases

- Combination of combinatorial search and heuristics to reach reasonable complexity
  - Search
    - Select a new variable assignment from several possibilities of assigning values to unassigned variables
  - Inference in CSPs (constraint propagation)
    - The process of determining how the constraints and the possible values of one variable affect the possible values of other variables
    - Enforcing local consistency in each part of the graph can cause inconsistent values to be eliminated
    - Using the graph structure can speed up the search process
      - e.g.,T is an independent sub-problem



Forward checking is only on simple form of constraint propagation

When a pair (X←v) is added to assignment A do:
  For each variable Y not in A do:
    For every constraint C relating Y to variables in A do:
      Remove all values from Y's domain that do not satisfy C

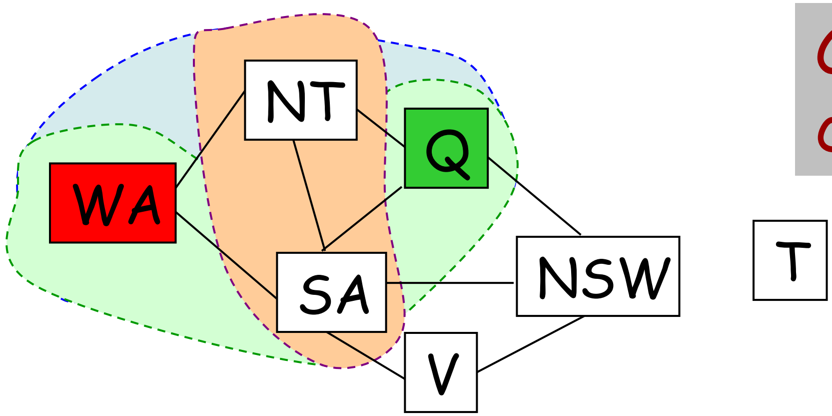


Forward Checking in Map Coloring

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

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

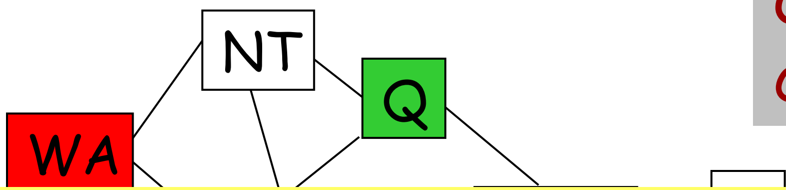
Forward Checking in Map Coloring



Contradiction that forward checking did not detect

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

Forward Checking in Map Coloring



Contradiction that forward checking did not detect

Detecting this contradiction requires a more powerful constraint propagation technique

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

Inference (constraint propagation) in CSPs

- Inference as a preprocessing stage
  - *AC-3 (arc consistency) algorithm*
    - Removes values from domains of variables (and propagates constraints) to provide all constrained pairs of variables arc consistent.
- Inference intertwined with search
  - Forward checking
    - When selecting a value for a variable, infers new domain reductions on neighboring unassigned variables
  - Maintaining Arc Consistency (MAC) - Constraint propagation
    - Forward checking + recursively propagating constraints when changes are made to the domains

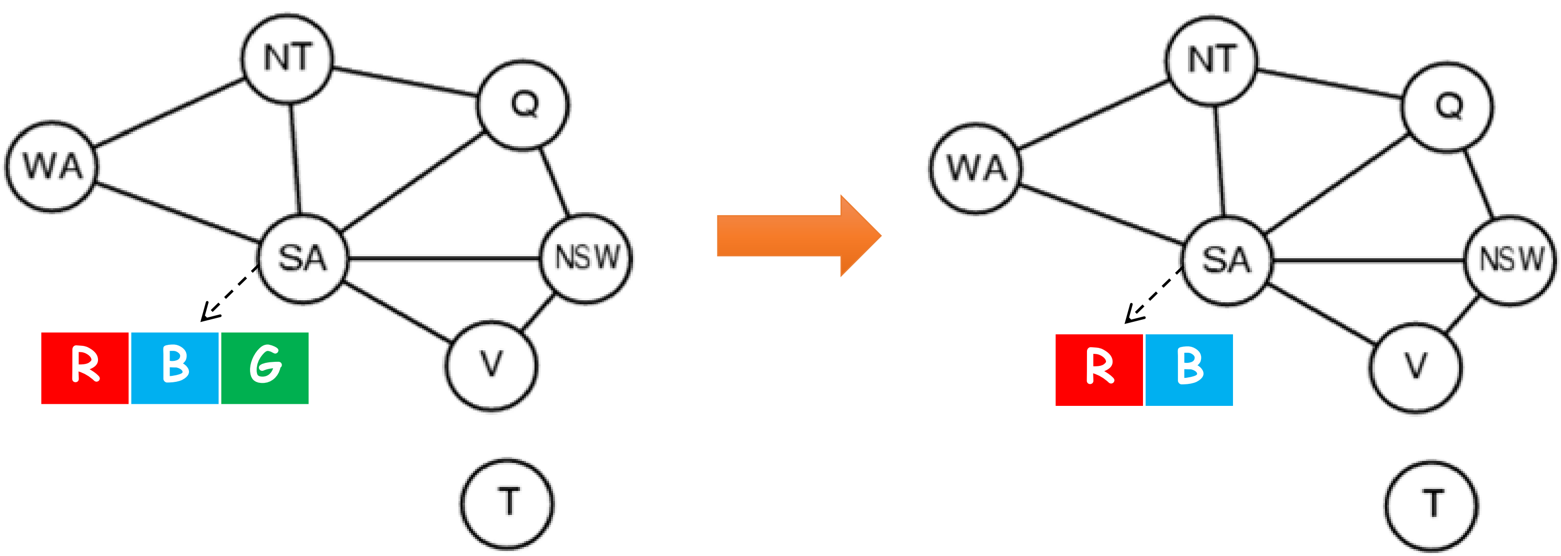
Local consistency

- Node consistency (1-consistency)
  - Unary constraints are satisfied
- Arc consistency (2-consistency)
  - Binary constraints are satisfied
- Path consistency
- k- consistency
- Global constraints

Node consistency

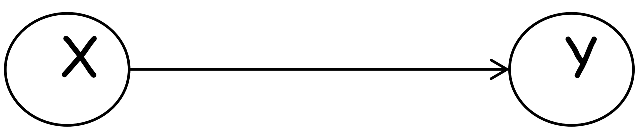
- A single variable (corresponding to a node in the CSP network) is node-consistent if all the values in the variable's domain satisfy the variable's unary constraints.

D_{SA}={red, blue, green} , SA$\neq$green $\rightarrow$ D_{SA}={red, blue}



Arc consistency

- A variable in a CSP is arc-consistent if every value in its domain satisfies the variable's binary constraints.
  - More formally, X_i is arc-consistent with respect to another variable X_j if for every value in the current domain D_i there is some value in the domain D_j that satisfies the binary constraint on the arc (X_i, X_j)

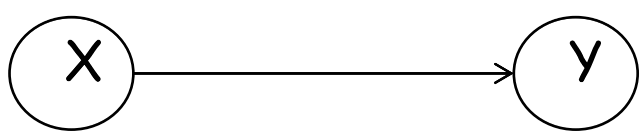


Constraint: $Y=X^2$

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

Arc consistency

- A variable in a CSP is arc-consistent if every value in its domain satisfies the variable's binary constraints.
  - More formally, X_i is arc-consistent with respect to another variable X_j if for every value in the current domain D_i there is some value in the domain D_j that satisfies the binary constraint on the arc (X_i, X_j)

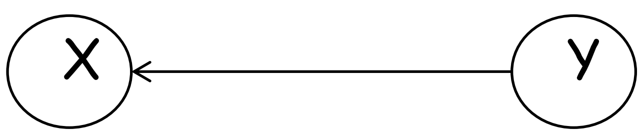


Constraint: $Y=X^2$

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

Arc consistency

- A variable in a CSP is arc-consistent if every value in its domain satisfies the variable's binary constraints.
  - More formally, X_i is arc-consistent with respect to another variable X_j if for every value in the current domain D_i there is some value in the domain D_j that satisfies the binary constraint on the arc (X_i, X_j)

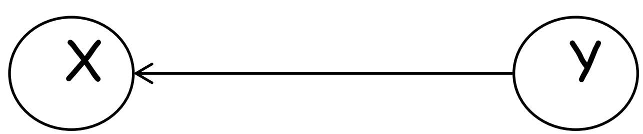


Constraint: $Y=X^2$

$\{0, 1, 2, 3, \cancel{4}, \cancel{5}, \cancel{6}, \cancel{7}, \cancel{8}, \cancel{9}\}$ $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Arc consistency

- A variable in a CSP is arc-consistent if every value in its domain satisfies the variable's binary constraints.
  - More formally, X_i is arc-consistent with respect to another variable X_j if for every value in the current domain D_i there is some value in the domain D_j that satisfies the binary constraint on the arc (X_i, X_j)



Constraint: $Y=X^2$

$\{0, 1, 2, 3, \cancel{4}, \cancel{5}, \cancel{6}, \cancel{7}, \cancel{8}, \cancel{9}\}$ $\{0, 1, \cancel{2}, \cancel{3}, 4, \cancel{5}, \cancel{6}, \cancel{7}, \cancel{8}, 9\}$

Constraint Propagation for Binary Constraints

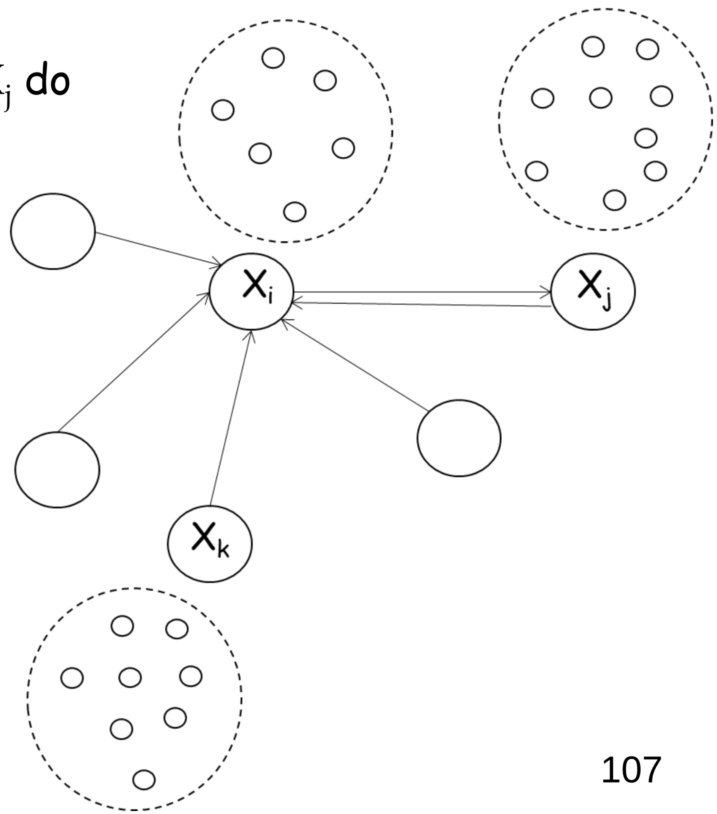
REMOVE-VALUES(X,Y)

1. removed ← false
2. For every value v in the domain of Y do
   - If there is no value u in the domain of X such that the constraint on (X,Y) is satisfied then
     a. Remove v from Y's domain
     b. removed ← true
3. Return removed

This algorithm makes Y arc-consistent with respect to X

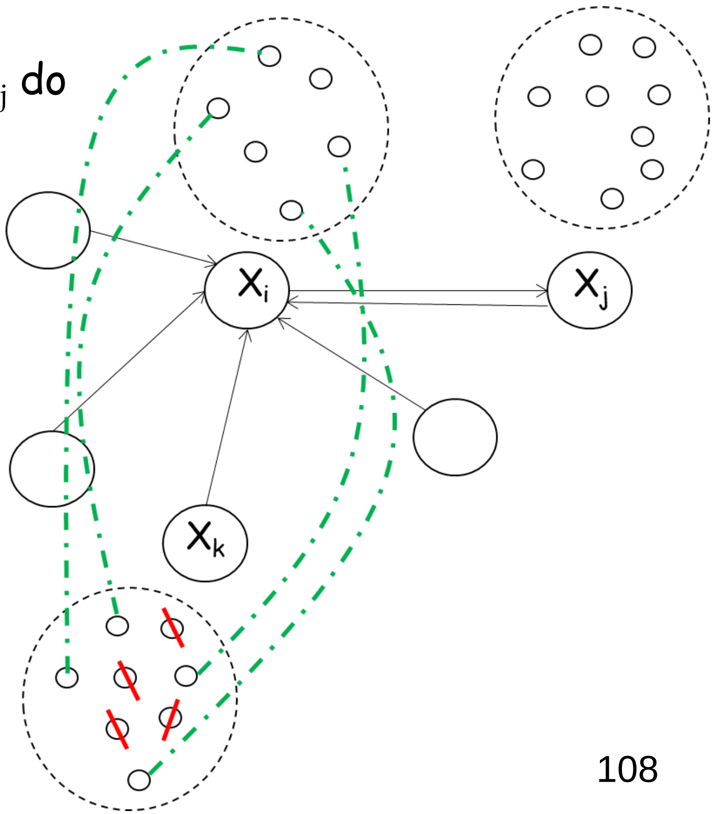
Arc-consistency algorithm AC-3

- For each arc (X_i, X_j) in the queue
  - Remove it from queue
  - Make X_i arc-consistent with respect to X_j
    1. If D_i remains unchanged then continue
    2. If $|D_i|=0$ then return false
    3. For each neighbor X_k of X_i except to X_j do
       - add (X_k, X_i) to queue



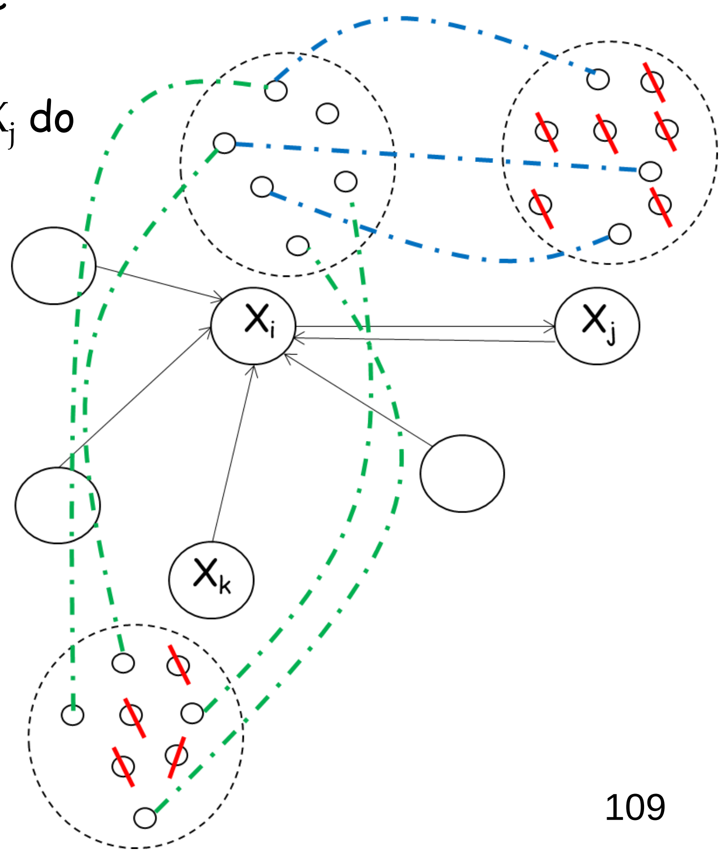
Arc-consistency algorithm AC-3

- For each arc (X_i, X_j) in the queue
  - Remove it from queue
  - Make X_i arc-consistent with respect to X_j
    1. If D_i remains unchanged then continue
    2. If $|D_i|=0$ then return false
    3. For each neighbor X_k of X_i except to X_j do
       - add (X_k, X_i) to queue



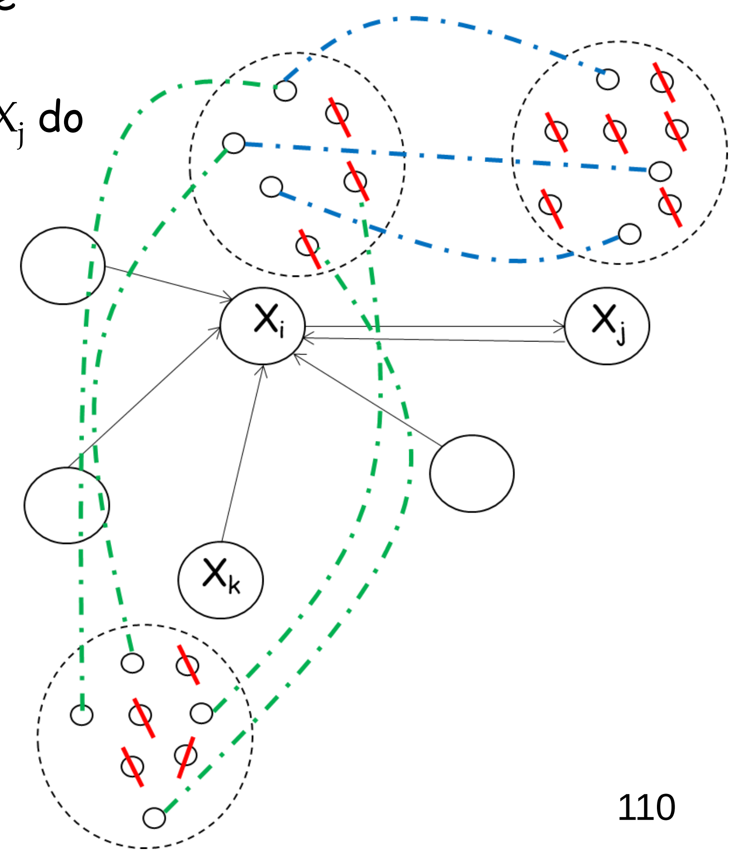
Arc-consistency algorithm AC-3

- For each arc (X_i, X_j) in the queue
  - Remove it from queue
  - Make X_i arc-consistent with respect to X_j
    1. If D_i remains unchanged then continue
    2. If $|D_i|=0$ then return false
    3. For each neighbor X_k of X_i except to X_j do
       - add (X_k, X_i) to queue



Arc-consistency algorithm AC-3

- For each arc (X_i, X_j) in the queue
  - Remove it from queue
  - Make X_i arc-consistent with respect to X_j
    1. If D_i remains unchanged then continue
    2. If $|D_i|=0$ then return false
    3. For each neighbor X_k of X_i except to X_j do
       - add (X_k, X_i) to queue



Arc-consistency algorithm AC-3

- For each arc (X_i, X_j) in the queue
  - Remove it from queue
  - Make X_i arc-consistent with respect to X_j
    1. If D_i remains unchanged then continue
    2. If $|D_i|=0$ then return false
    3. For each neighbor X_k of X_i except to X_j do
       - add (X_k, X_i) to queue
- Removing a value from a domain may cause further inconsistency, so we have to repeat the procedure until everything is consistent.
- When queue is empty, resulted CSP is equivalent to the original CSP
  - Same solution (usually reduced domains speed up the search)

Arc-consistency algorithm AC-3

function AC-3(*csp*) **returns** false if an inconsistency is found and true otherwise
  inputs: *csp*, a binary CSP with components $(X,\ D,\ C)$
  local variables: *queue*, a queue of arcs, initially all the arcs in *csp*

  while *queue* is not empty **do**
    $(X_i,\ X_j) \leftarrow$ REMOVE-FIRST(*queue*)
    if REVISE(*csp*, $X_i,\ X_j$) **then**
      if size of $D_i\ =\ 0$ **then return** *false*
      for each X_k **in** X_i.NEIGHBORS - $\{X_j\}$ **do**
        add $(X_k,\ X_i)$ to *queue*
  return *true*

---

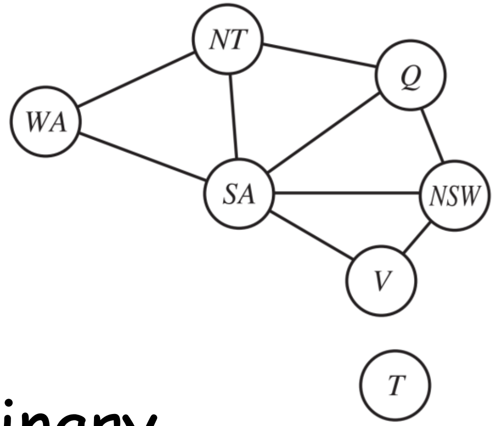
function REVISE(*csp*, $X_i,\ X_j$) **returns** true iff we revise the domain of X_i
  revised $\leftarrow$ *false*
  for each x **in** D_i **do**
    if no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j **then**
      delete x from D_i
      revised $\leftarrow$ *true*
  return *revised*

Complexity Analysis of AC3

- n = number of variables
- d = size of initial domains
- c = number of binary constraints
- Each arc (X_k, X_i) is inserted in queue up to d times
- Checking consistency of an arc takes $O(d^2)$ time
- AC3 takes $O(d \times c \times d^2) = O(c \times d^3)$ time
- Usually more expensive than forward checking

Arc consistency: map coloring example

- For general map coloring problem all pairs of variables are arc-consistent if $|D_i| \geq 2$ $(i=1,\ldots,n)$
  - Arc consistency can do nothing in map coloring with two colors
- We need stronger notion of consistency to detect failure at start.

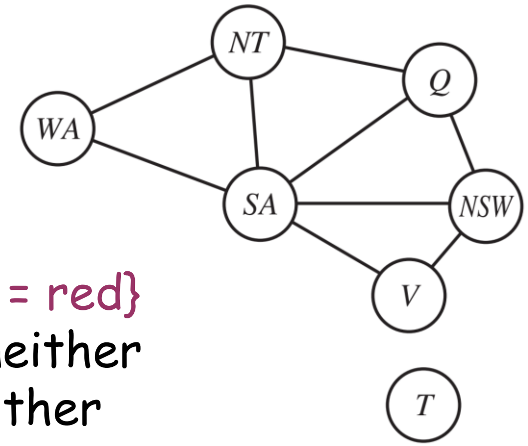


- Path consistency tightens the binary constraints by using implicit constraints that are inferred by looking at triples of variables

Path consistency

- A two-variable set $\{X_i, X_j\}$ is path-consistent with respect to a third variable X_m if, for every assignment $\{X_i = a, X_j = b\}$ consistent with the constraints on $\{X_i, X_j\}$, there is an assignment to X_m that satisfies the constraints on $\{X_i, X_m\}$ and $\{X_m, X_j\}$.
- This is called path consistency because one can think of it as looking at a path from X_i to X_j with X_m in the middle.

- Example
  - Make the set {WA,SA} path consistent with respect to NT
    - There are only two assignments:
      {WA = red, SA = blue} and {WA = blue, SA = red}
    - With both of these assignments NT can be neither red nor blue (because it would conflict with either WA or SA)



k-consistency

- A CSP is k-consistent if, for any set of k–1 variables and for any consistent assignment to those variables, a consistent value can always be assigned to any kth variable.
  - 1-consistency
    - Given the empty set, we can make any set of one variable consistent.
    - Node consistency
  - 2-consistency
    - Arc consistency
  - 3-consistency
    - Path consistency (For binary constraint networks)

Strongly k-consistent

- A CSP is strongly k-consistent if it is k-consistent and is also (k – 1)-consistent, (k – 2)-consistent, . . . all the way down to 1-consistent.
- Suppose we have a CSP with n nodes and make it strongly n-consistent (i.e., strongly k-consistent for k=n)
  - First, we choose a consistent value for X_1
  - We are then guaranteed to be able to choose a value for X_2, X_3 and so on.
  - For each variable X_i, we need only search through the d values in the domain to find a value consistent with $X_1, \ldots, X_{i-1}$.
  - We are guaranteed to find a solution in time $O(n^2d)$.

X_1	0
X_2	d
X_3	2d
⋮	⋮
X_n	(n-1)d

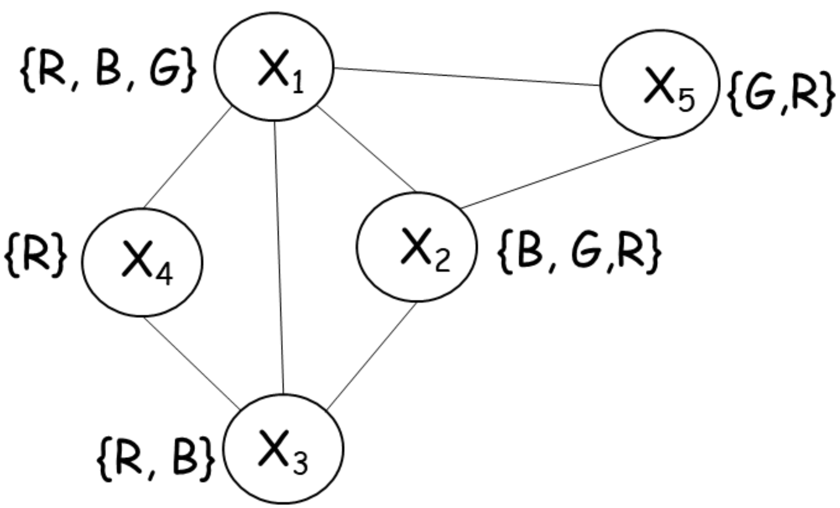
Which level of consistency?

- Of course, there is no free lunch
  - Any algorithm for establishing n-consistency must take time exponential in n in the worst case.
  - Also requires space that is exponential in n
    - The memory issue is even more severe than the time
- Determining the appropriate level of consistency checking is mostly an empirical science
  - Practitioners commonly compute 2-consistency and less commonly 3-consistency

Global constraints

- Global constraints occur frequently in real problems and can be handled by special-purpose algorithms that are more efficient than the general-purpose methods described so far.
- Alldiff
  - If m variables are involved in the constraint, and if they have n possible distinct values altogether, and m > n, then the constraint cannot be satisfied
  - A simple algorithm to apply Alldiff constraint

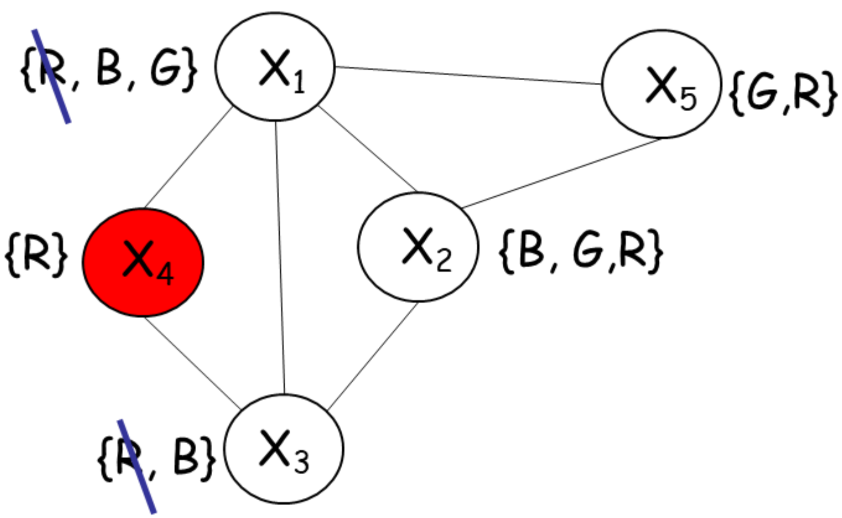
1. remove any variable in the constraint that has a singleton domain, and delete that variable's value from the domains of the remaining variables
2. Repeat as long as there are singleton variables
3. If at any point an empty domain is produced or there are more variables than domain values left, then an inconsistency has been detected



Global constraints

- Global constraints occur frequently in real problems and can be handled by special-purpose algorithms that are more efficient than the general-purpose methods described so far.
- Alldiff
  - If m variables are involved in the constraint, and if they have n possible distinct values altogether, and m > n, then the constraint cannot be satisfied
  - A simple algorithm to apply Alldiff constraint

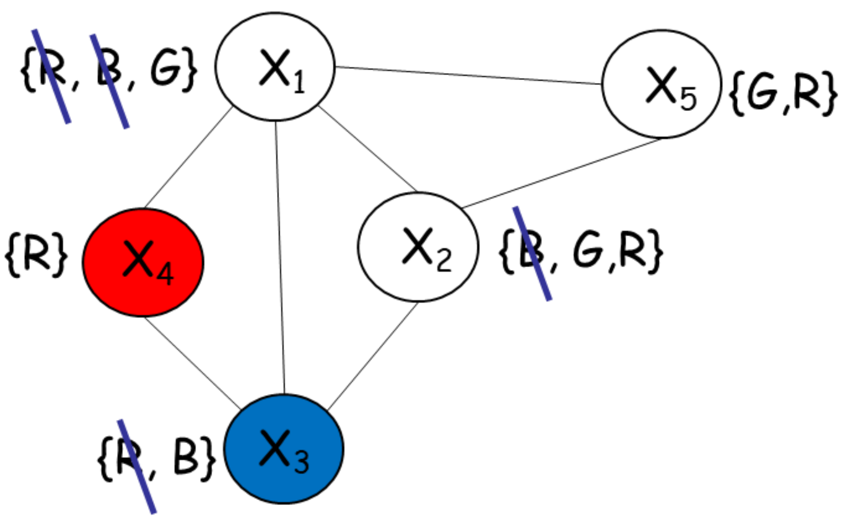
1. remove any variable in the constraint that has a singleton domain, and delete that variable's value from the domains of the remaining variables
2. Repeat as long as there are singleton variables
3. If at any point an empty domain is produced or there are more variables than domain values left, then an inconsistency has been detected



Global constraints

- Global constraints occur frequently in real problems and can be handled by special-purpose algorithms that are more efficient than the general-purpose methods described so far.
- Alldiff
  - If m variables are involved in the constraint, and if they have n possible distinct values altogether, and m > n, then the constraint cannot be satisfied
  - A simple algorithm to apply Alldiff constraint

1. remove any variable in the constraint that has a singleton domain, and delete that variable's value from the domains of the remaining variables
2. Repeat as long as there are singleton variables
3. If at any point an empty domain is produced or there are more variables than domain values left, then an inconsistency has been detected



Global constraints

- Resource constraint (sometimes called the atmost constraint)
- Example
  - In a scheduling problem, let $P_1, \ldots, P_4$ denote the numbers of personnel assigned to each of four tasks
  - The constraint that no more than 10 personnel are assigned in total is written as $Atmost(10, P_1, P_2, P_3, P_4)$
  - We can detect an inconsistency simply by checking the sum of the minimum values of the current domains
    - If each variable has the domain {3, 4, 5, 6}, the Atmost constraint cannot be satisfied
  - We can also enforce consistency by deleting the maximum value of any domain if it is not consistent with the minimum values of the other domains
    - If each variable in our example has the domain {2, 3, 4, 5, 6}, the values 5 and 6 can be deleted from each domain.

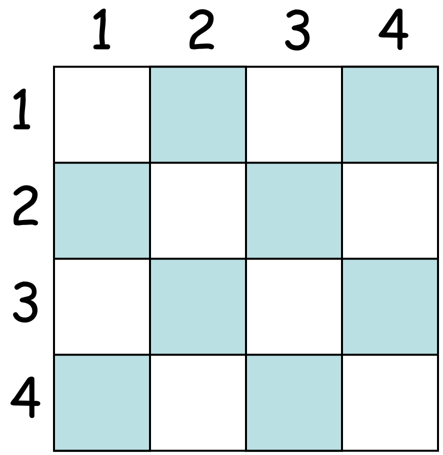
Global constraints

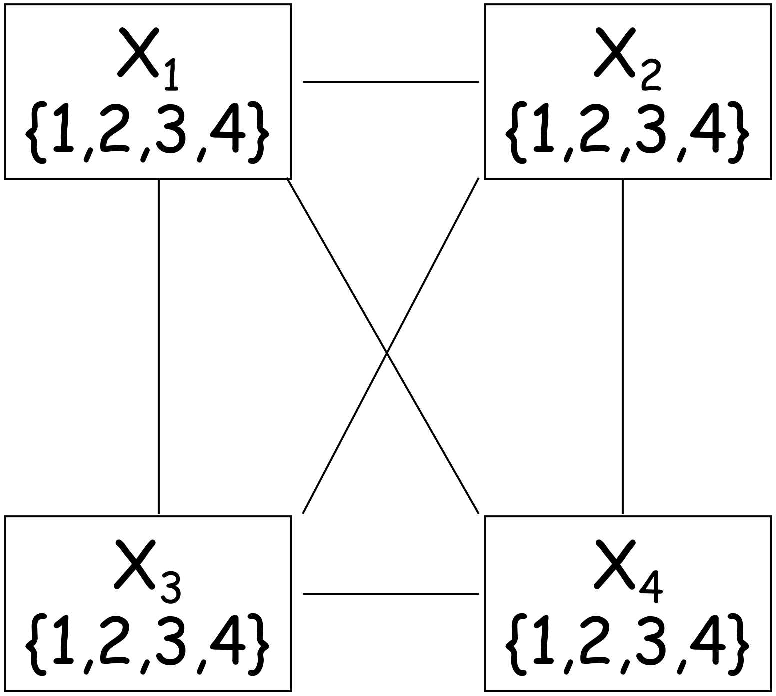
- Bounds constraint
  - Sometimes domains are represented by upper and lower bounds and are managed by bounds propagation
- Example
  - In an airline-scheduling problem,
    - Suppose there are two flights, F_1 and F_2, for which the planes have capacities 165 and 385, respectively.
    - The initial domains for the numbers of passengers on each flight are:

      $D_1 = [0, 165] \qquad D_2 = [0, 385]$
    - There is an additional constraint that the two flights together must carry 420 people: $F1 + F2 = 420$.
    - Propagating bounds constraints, we reduce the domains to

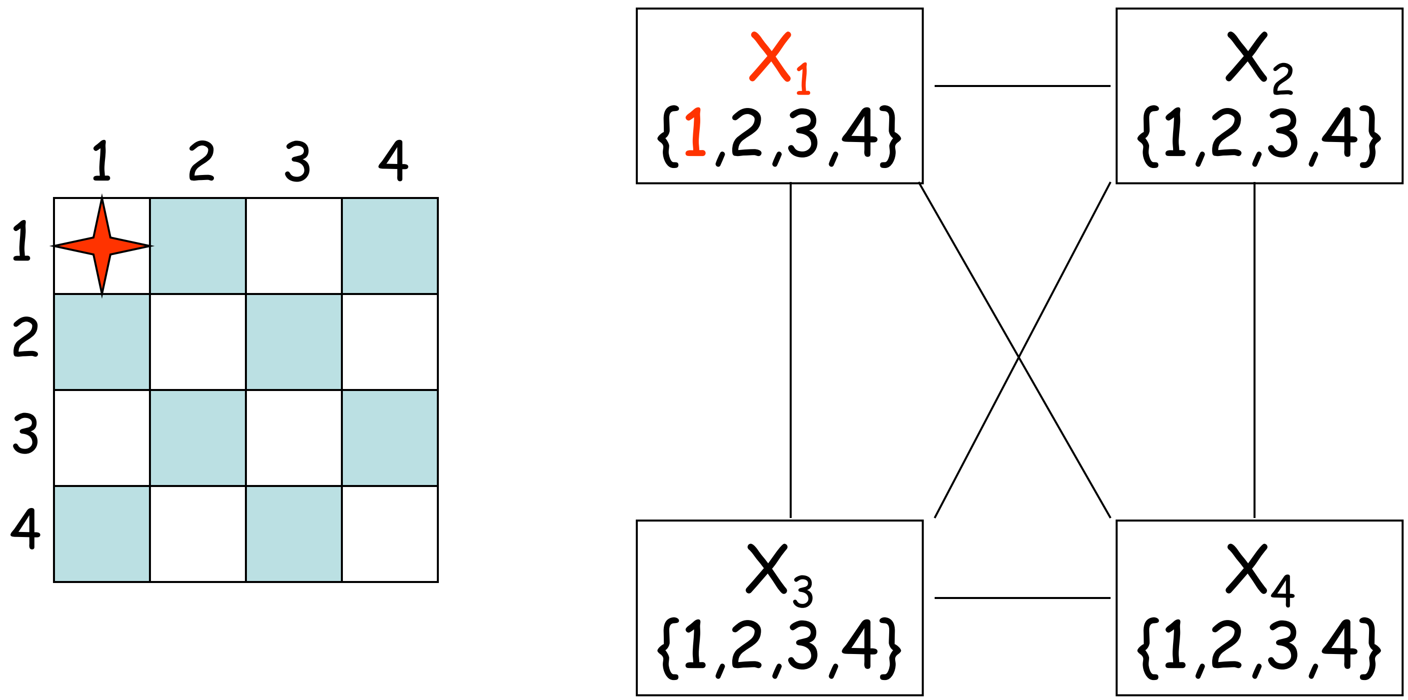
      $D_1 = [255, 385] \qquad D_2 = [35, 165]$

4-Queens Problem (FC+MRV)

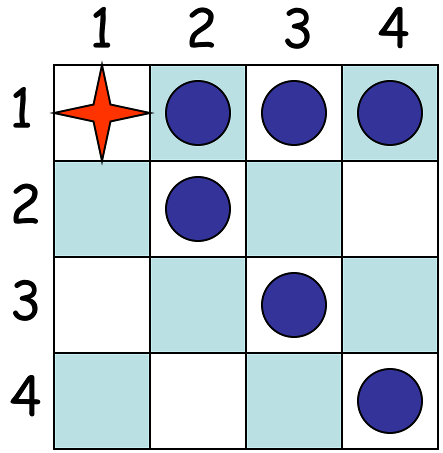


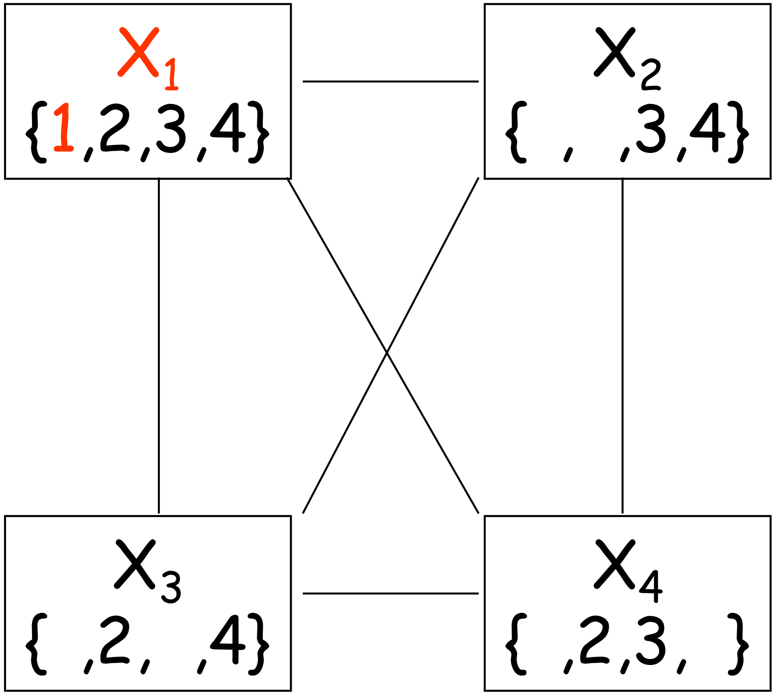


4-Queens Problem (FC+MRV)

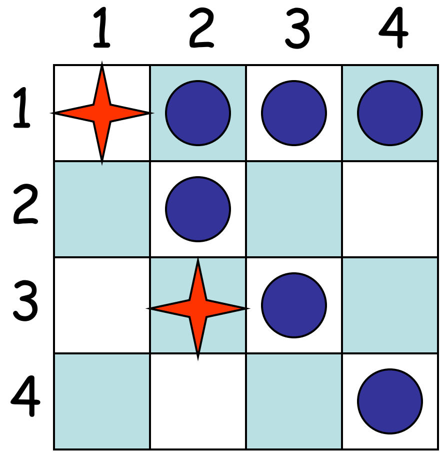


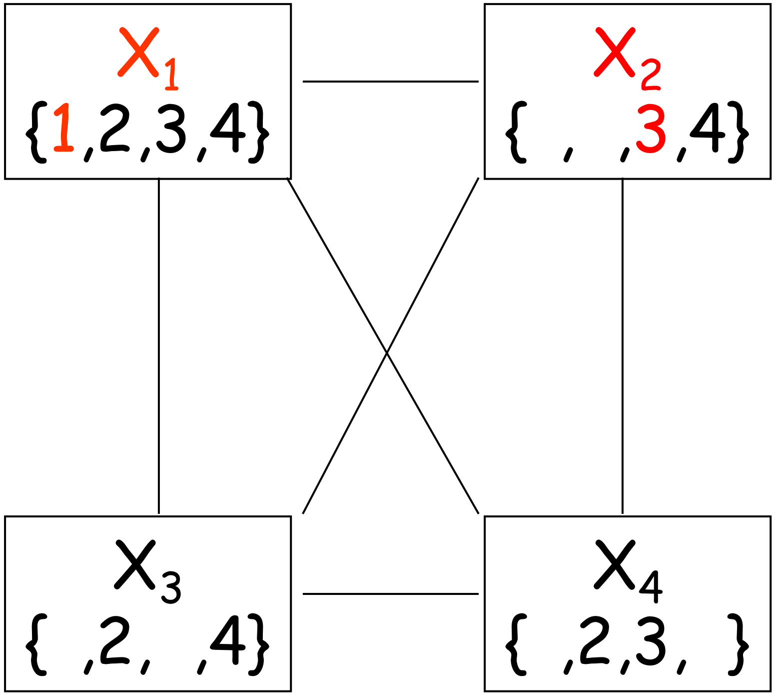
4-Queens Problem (FC+MRV)



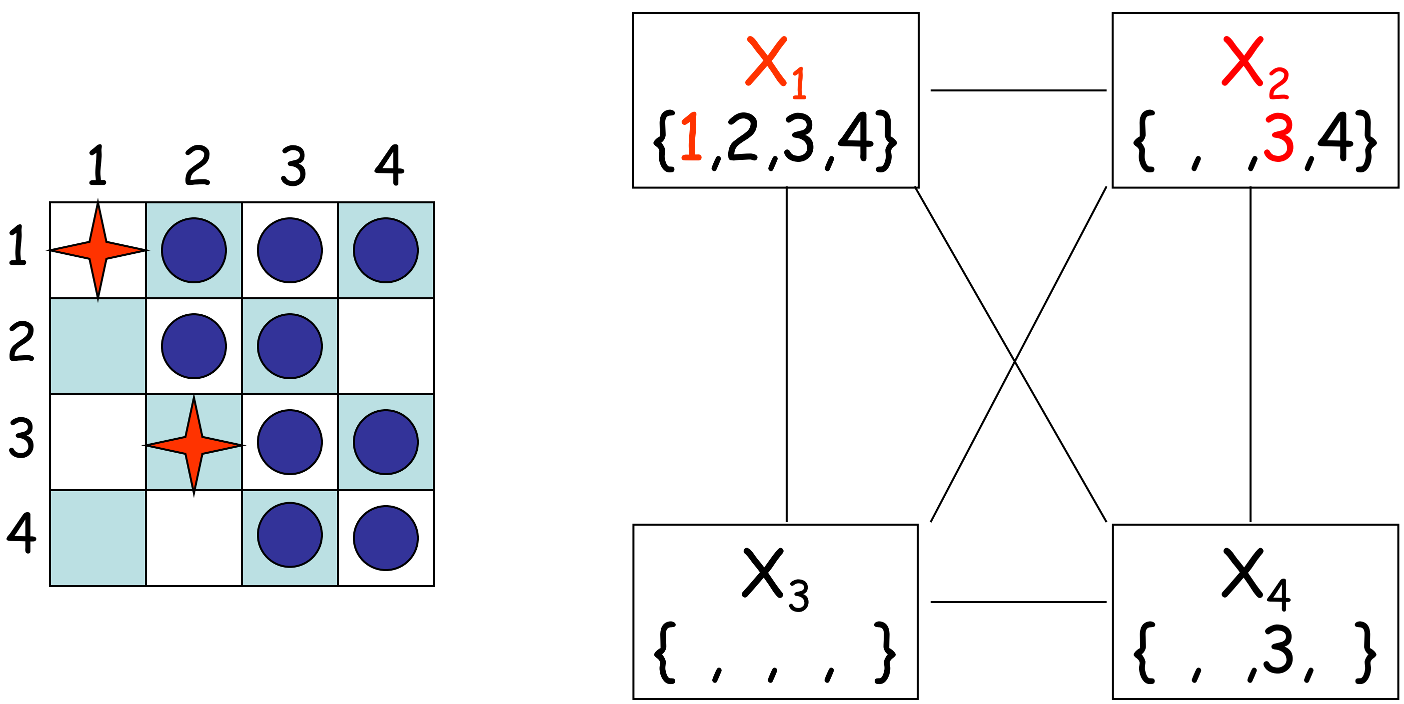


4-Queens Problem (FC+MRV)

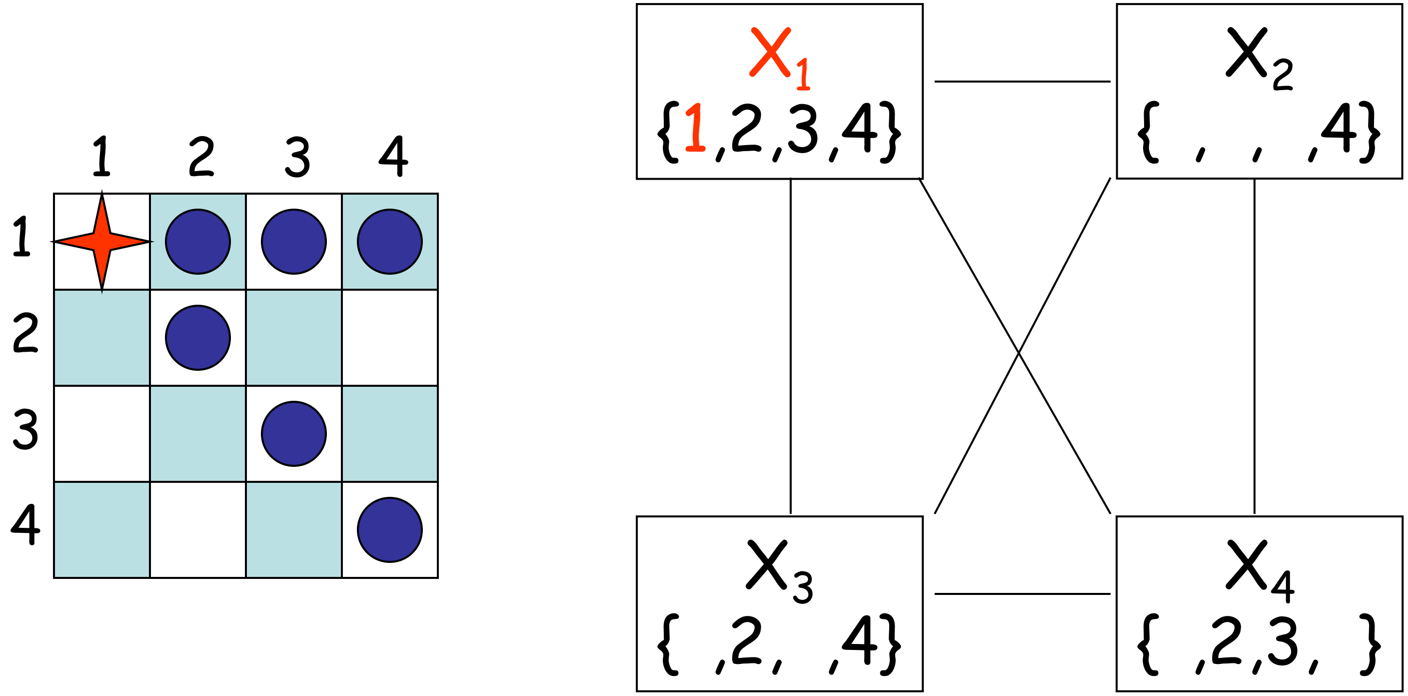




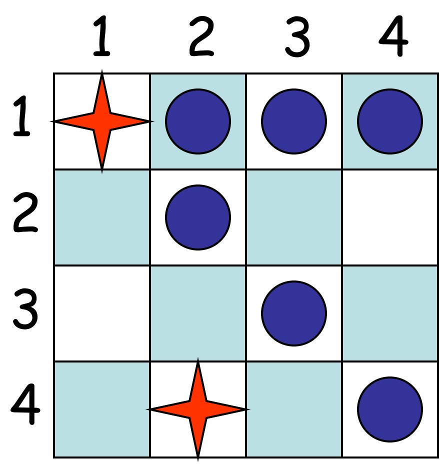
4-Queens Problem (FC+MRV)

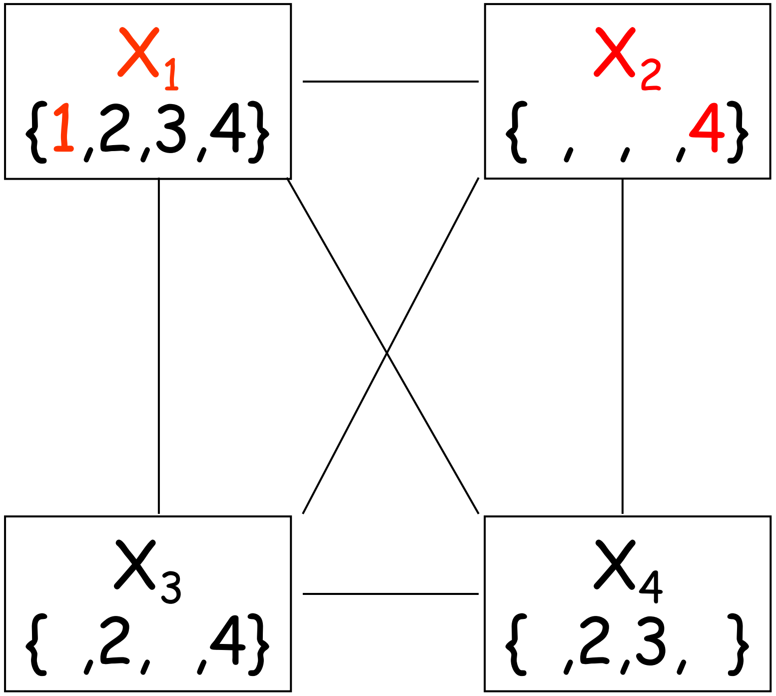


4-Queens Problem (FC+MRV)

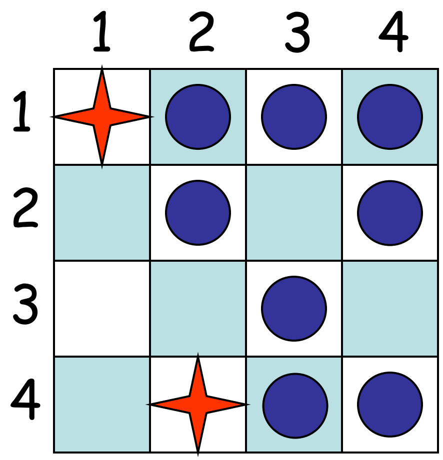


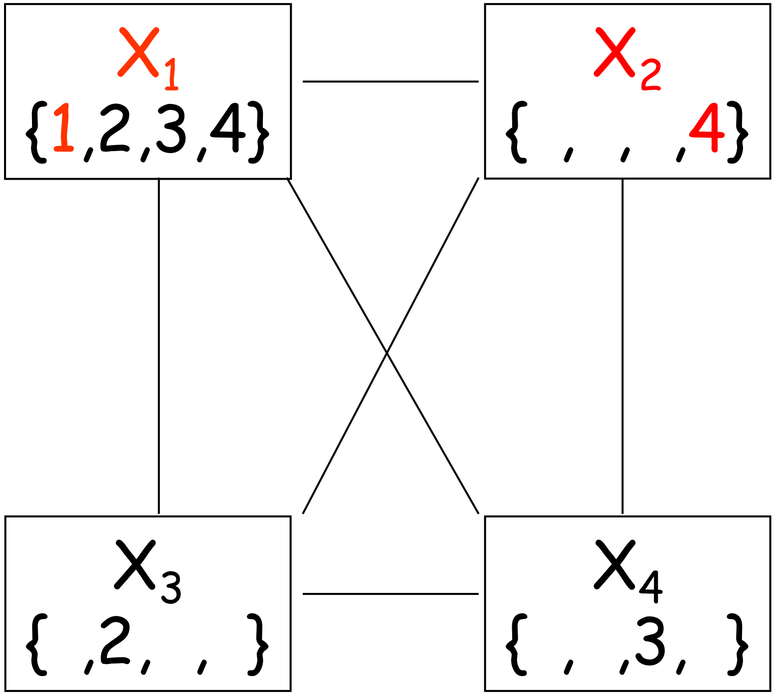
4-Queens Problem (FC+MRV)



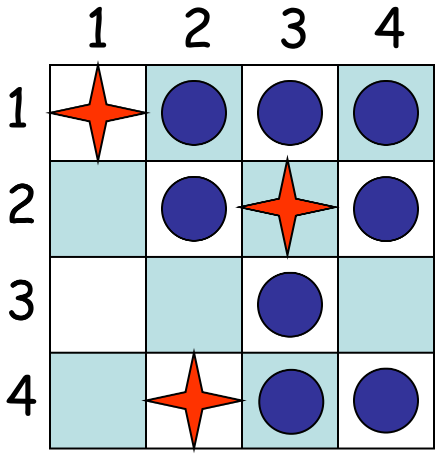


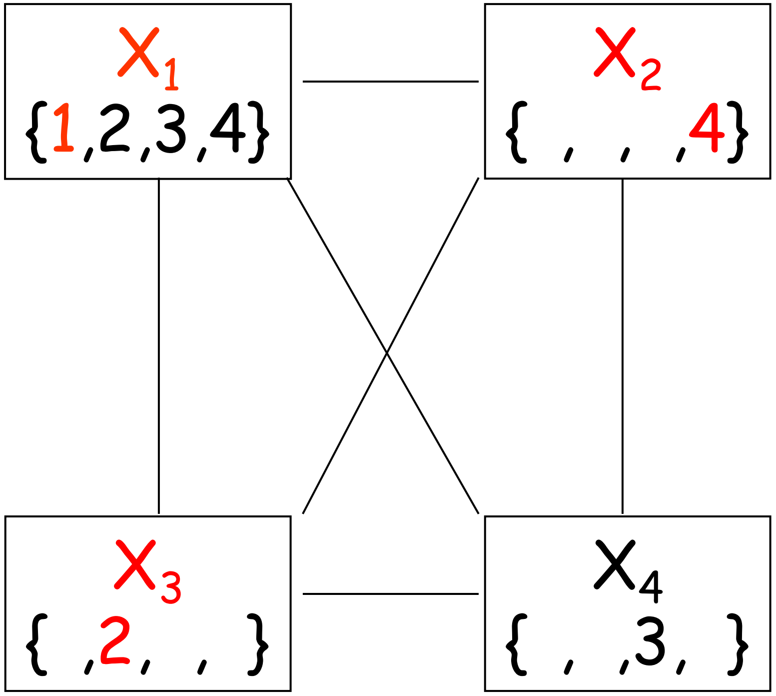
4-Queens Problem (FC+MRV)



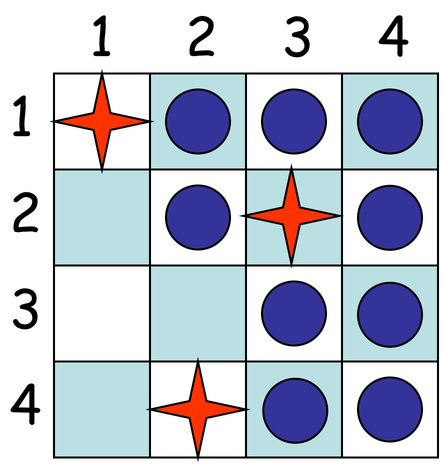


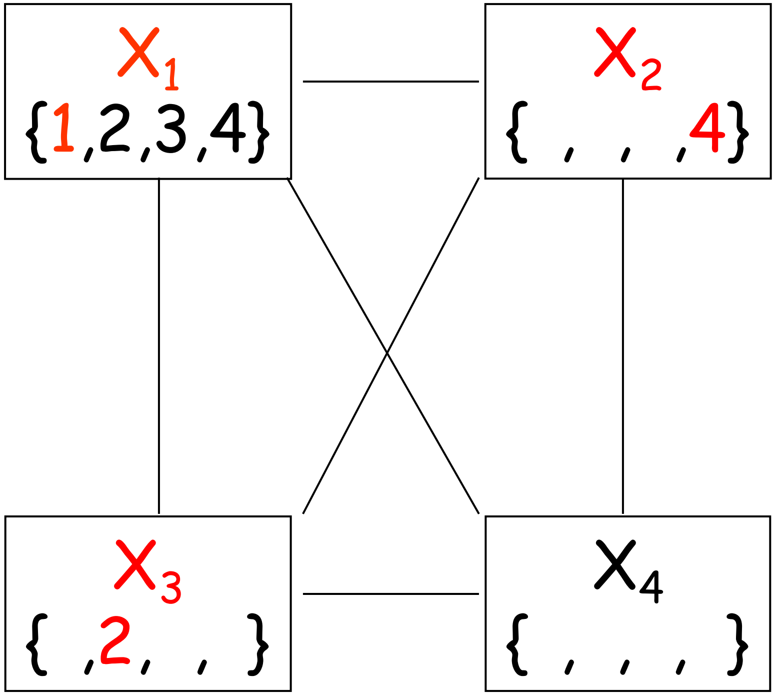
4-Queens Problem (FC+MRV)



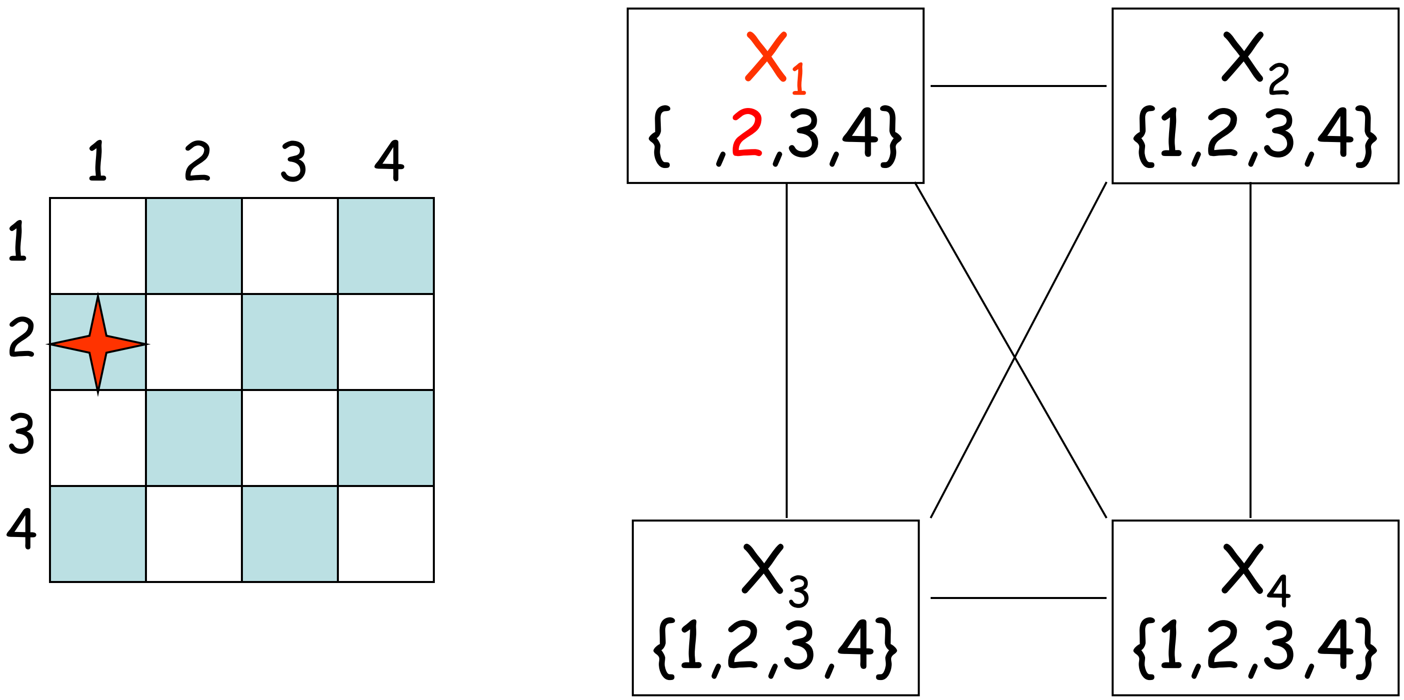


4-Queens Problem (FC+MRV)





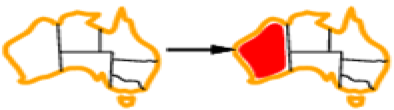
4-Queens Problem (FC+MRV)

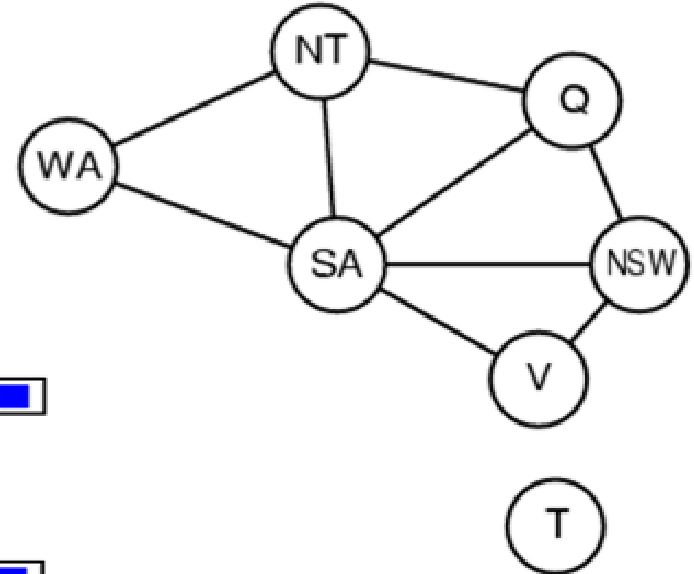


Constraint propagation

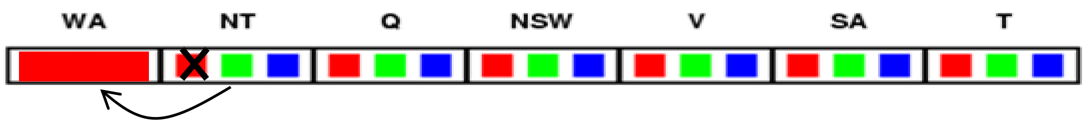
- Although forward checking detects many inconsistencies, it does not detect all of them.
- Maintaining Arc Consistency (MAC)
  - After a variable X_i is assigned a value, the INFERENCE procedure calls AC-3
  - Instead of a queue of all arcs in the CSP, we start with only the arcs (X_j, X_i) for all X_j that are unassigned variables that are neighbors of X_i.

Example

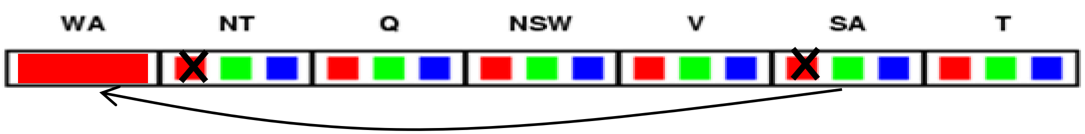




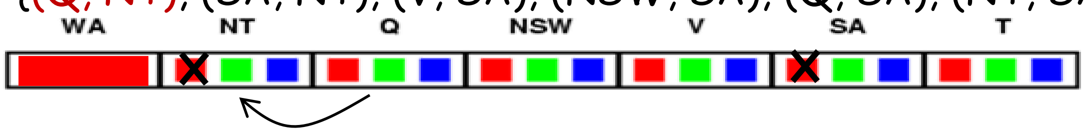
{(NT, WA) ,(SA, WA)}



{(SA, WA), (Q, NT), (SA, NT)}

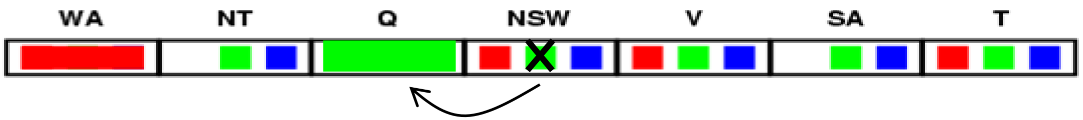


{(Q, NT), (SA, NT), (V, SA), (NSW, SA), (Q, SA), (NT, SA)}

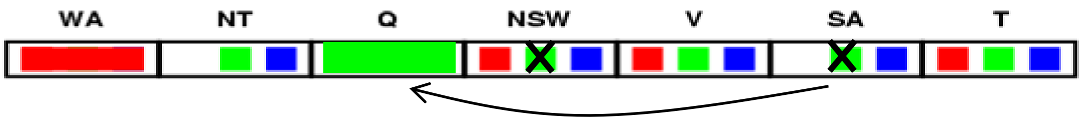


Example

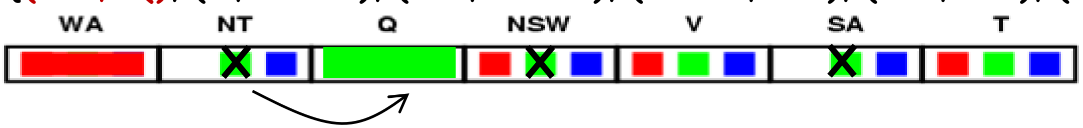
{(NSW, Q) ,(SA, Q), (NT, Q)}



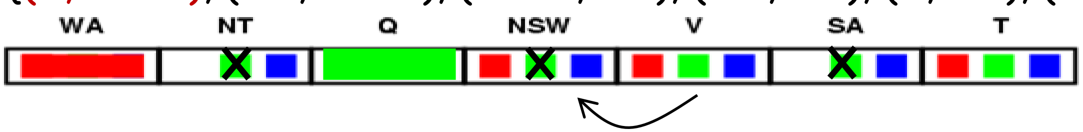
{(SA, Q), (NT, Q), (V, NSW), (SA, NSW)}

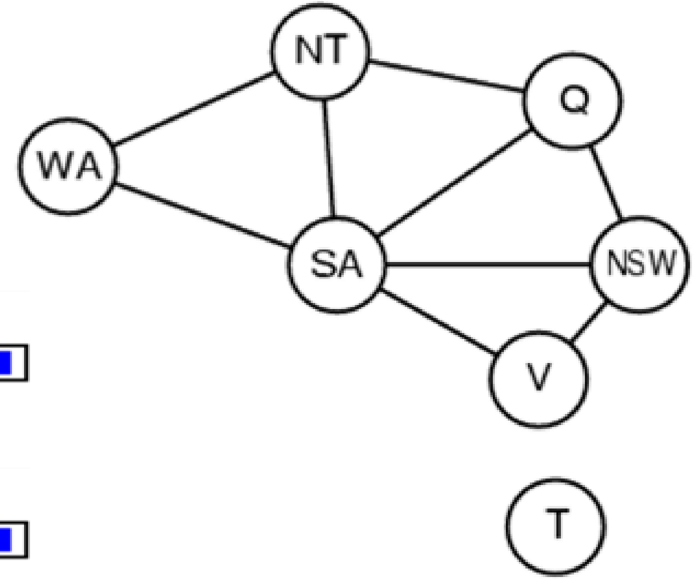


{(NT, Q), (V, NSW), (SA, NSW), (NSW, SA), (NT, SA), (V, SA)}

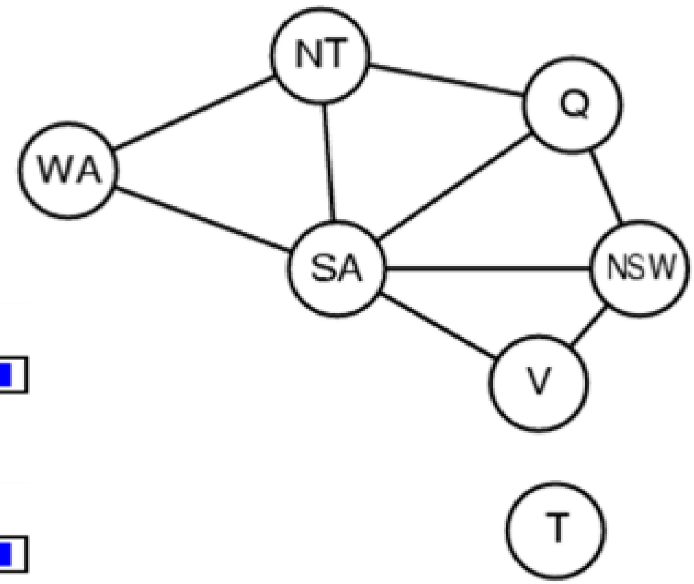


{(V, NSW), (SA, NSW), (NSW, SA), (NT, SA), (V, SA), (SA, NT)}

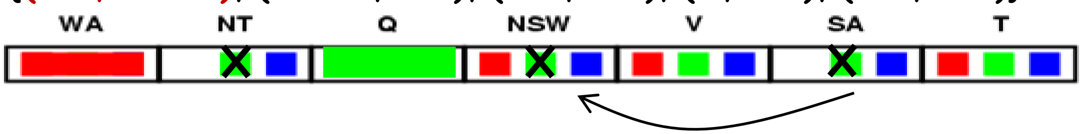




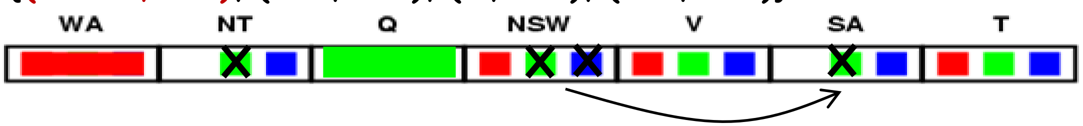
Example



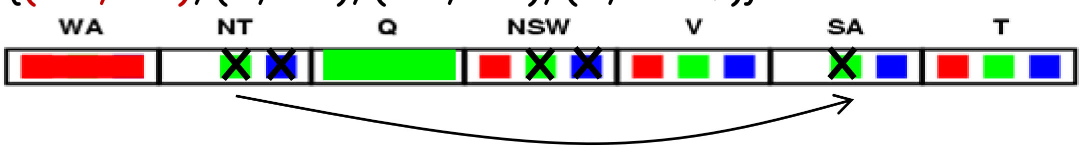
{(SA, NSW), (NSW, SA), (NT, SA), (V, SA), (SA, NT)}



{(NSW, SA), (NT, SA), (V, SA), (SA, NT)}



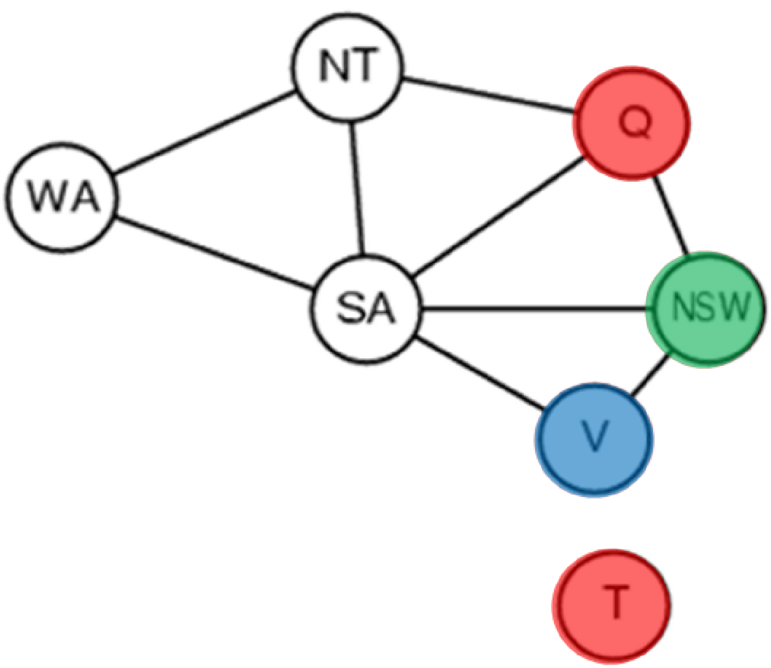
{(NT, SA), (V, SA), (SA, NT), (V, NSW)}



Simple Backtracking

- The BACKTRACKING-SEARCH algorithm, when a branch of the search fails, backs up to the preceding variable and tries a different value for it.
  - Consider what happens when we apply simple backtracking with a fixed variable ordering:

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

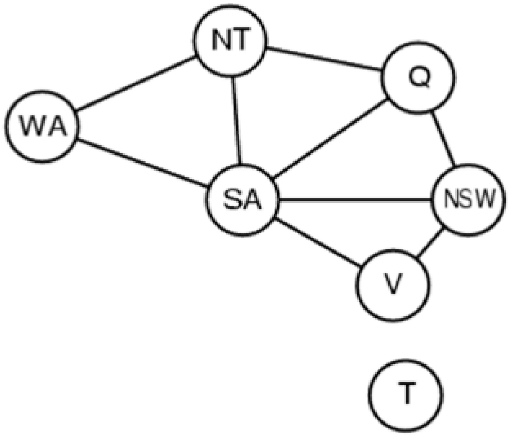


Back-jumping

- A more intelligent approach to backtracking is to backtrack to a variable that might fix the problem
  - A variable that was responsible for making one of the possible values of SA impossible.
- conflict set of X
  - A set of assignments that are in conflict with some value for X.
- The **back-jumping** method backtracks to the most recent assignment in the conflict set.

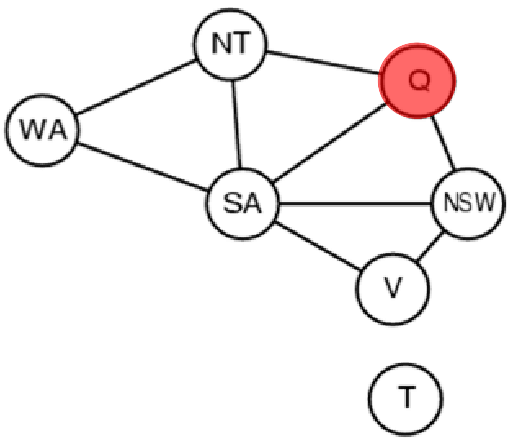
Back-jumping: Example

Suppose we have generated the partial assignment
{Q = red, NSW = green, V = blue, T = red}



Back-jumping: Example

Suppose we have generated the partial assignment
{Q = red, NSW = green, V = blue, T = red}

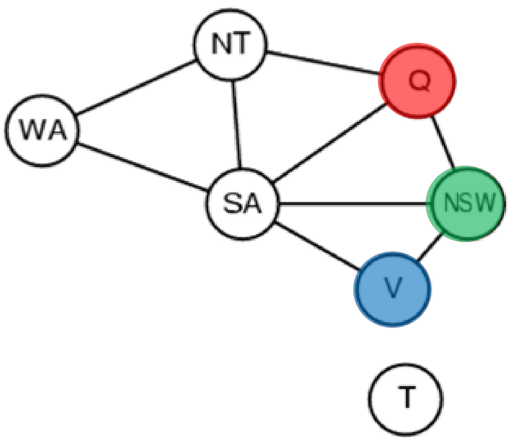


Q=red

Conf(Q)={}

Back-jumping: Example

Suppose we have generated the partial assignment
{Q = red, NSW = green, V = blue, T = red}



Q=red → NSW=green → V=blue

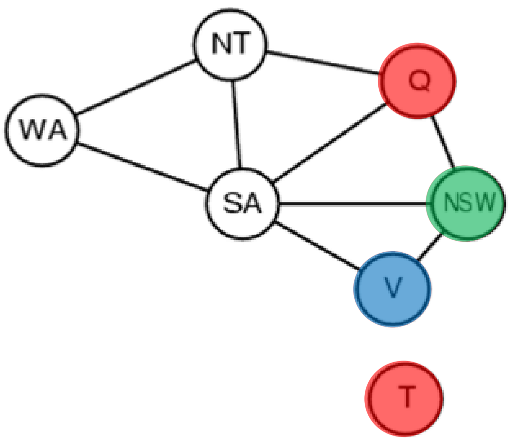
Conf(Q)={}

Conf(NSW)={Q}

Conf(V)={NSW}

Back-jumping: Example

Suppose we have generated the partial assignment
{Q = red, NSW = green, V = blue, T = red}



Q=red → NSW=green → V=blue → T=red

Conf(Q)={}

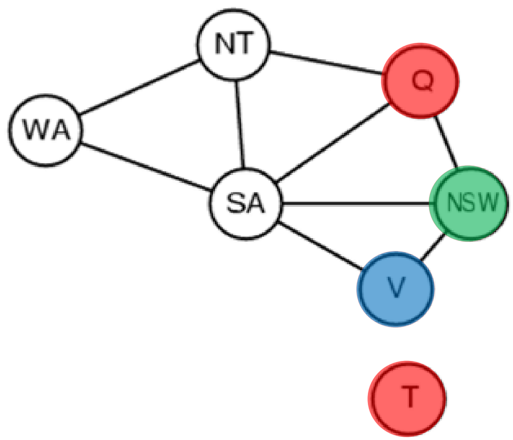
Conf(NSW)={Q}

Conf(V)={NSW}

Conf(T)={}

Back-jumping: Example

Suppose we have generated the partial assignment
{Q = red, NSW = green, V = blue, T = red}



Q=red → NSW=green → V=blue → T=red → SA=?

Conf(Q)={}

Conf(NSW)={Q}

Conf(V)={NSW}

Conf(T)={}

Conf(SA)={Q,NSW,V}

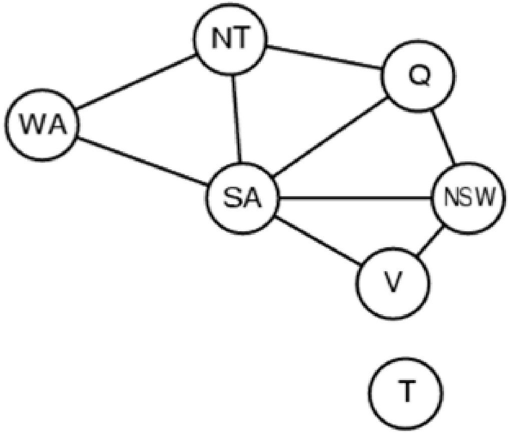
Back-jumping would jump over Tasmania and try a new value for V

Back-jumping

- Back-jumping occurs when every value in a domain is in conflict with the current assignment; but forward checking detects this event and prevents the search from ever reaching such a node!
  - In fact, it can be shown that every branch pruned by back-jumping is also pruned by forward checking.
- Hence, simple back-jumping is redundant in a forward-checking search or, indeed, in a search that uses stronger consistency checking, such as MAC.

Back-jumping: Example

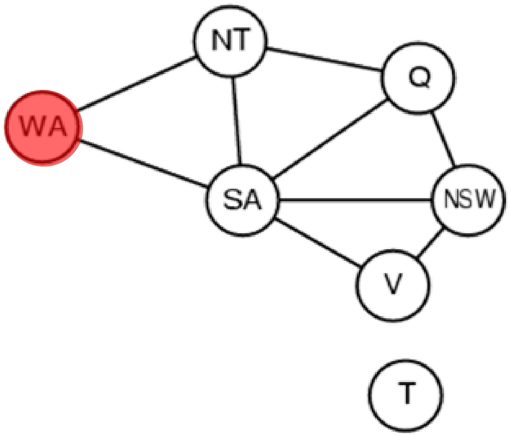
Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?



Back-jumping: Example

Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?

WA=red Conf(WA)={}



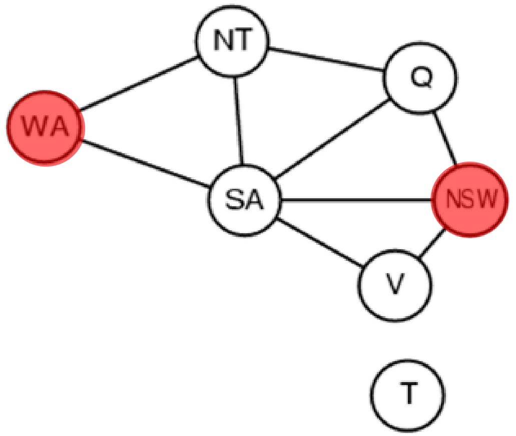
Back-jumping: Example

Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?

WA=red   Conf(WA)={}

↓

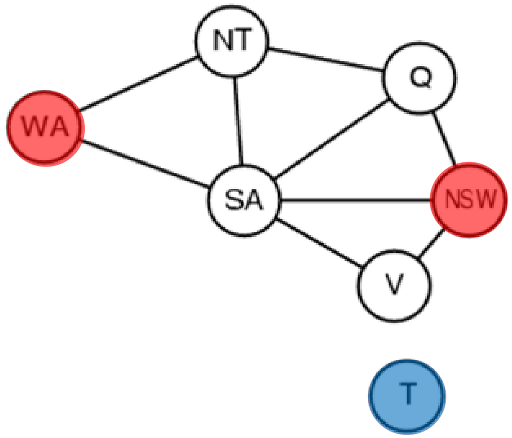
NSW=red   Conf(NSW)={}



Back-jumping: Example

Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?

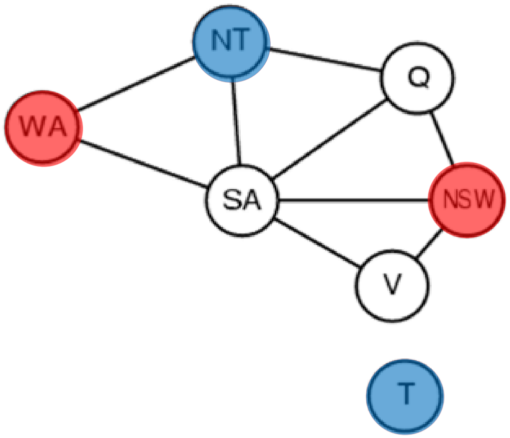
WA=red Conf(WA)={}
↓
NSW=red Conf(NSW)={}
↓
T=blue Conf(T)={}



Back-jumping: Example

Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?

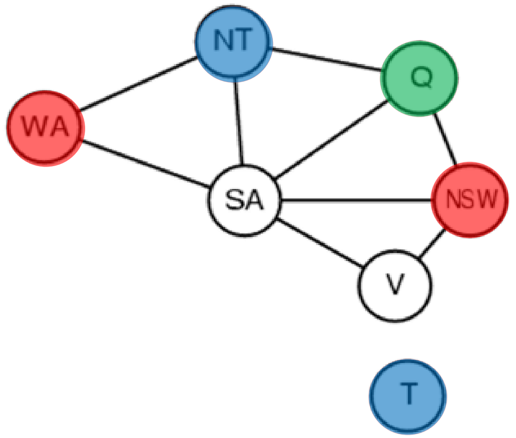
WA=red	Conf(WA)={}
↓	
NSW=red	Conf(NSW)={}
↓	
T=blue	Conf(T)={}
↓	
NT=blue	Conf(NT)={WA}



Back-jumping: Example

Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?

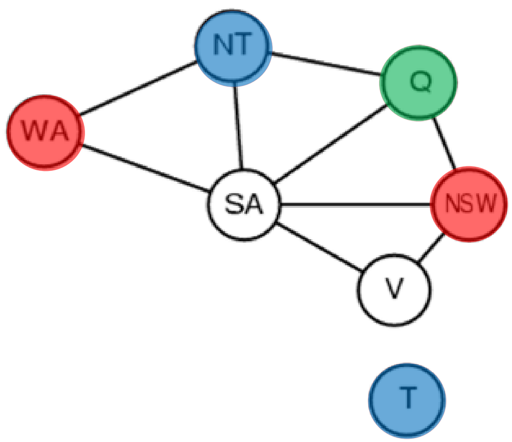
WA=red	Conf(WA)={}
↓	
NSW=red	Conf(NSW)={}
↓	
T=blue	Conf(T)={}
↓	
NT=blue	Conf(NT)={WA}
↓	
Q=green	Conf(Q)={NSW, NT}



Back-jumping: Example

Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?

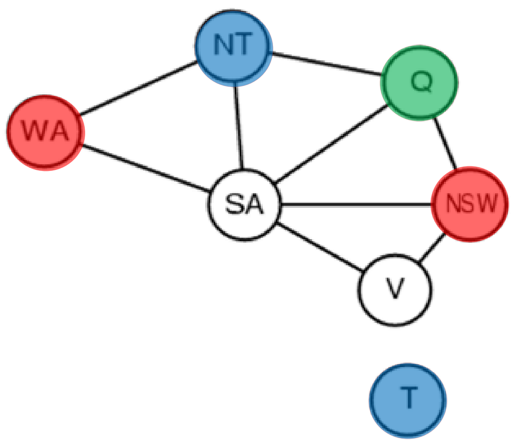
WA=red ↓	Conf(WA)={}
NSW=red ↓	Conf(NSW)={}
T=blue ↓	Conf(T)={}
NT=blue ↓	Conf(NT)={WA}
Q=green ↓	Conf(Q)={NSW, NT}
SA=?	Conf(SA)={WA, NSW, NT, Q}



Back-jumping: Example

Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?

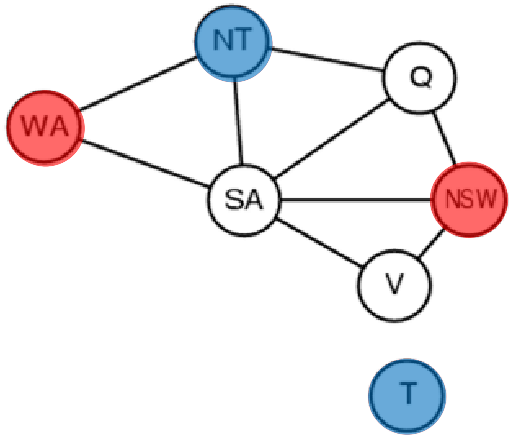
WA=red	Conf(WA)={}
↓	
NSW=red	Conf(NSW)={}
↓	
T=blue	Conf(T)={}
↓	
NT=blue	Conf(NT)={WA}
↓	
Q=green	Conf(Q)={NSW, NT}
↓	
SA=?	Conf(SA)={WA, NSW, NT, Q}



Back-jumping: Example

Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?

WA=red	Conf(WA)={}
NSW=red	Conf(NSW)={}
T=blue	Conf(T)={}
NT=blue	Conf(NT)={WA}
Q=green	Conf(Q)={NSW, NT}
SA=?	Conf(SA)={WA, NSW, NT, Q}



Back-jumping: Example

Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?

WA=red — Conf(WA)={}

↓

NSW=red — Conf(NSW)={}

↓

T=blue — Conf(T)={}

↓

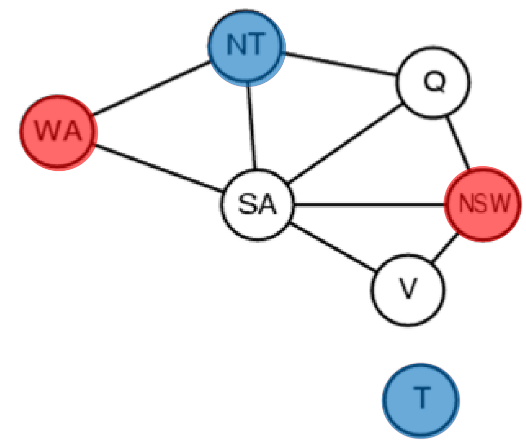
NT=blue — Conf(NT)={WA}

↓

Q=green — Conf(Q)={NSW, NT}

↓

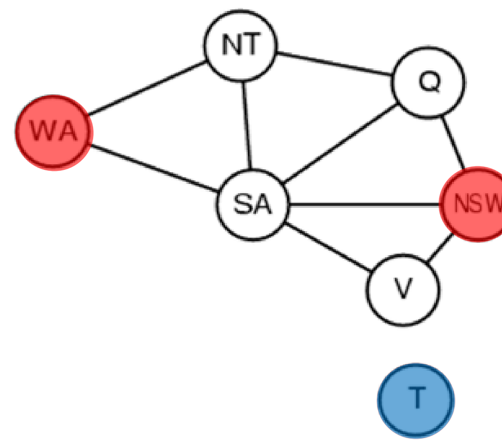
SA=? — Conf(SA)={WA, NSW, NT, Q}



Back-jumping: Example

Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?

WA=red	Conf(WA)={}
NSW=red	Conf(NSW)={}
T=blue	Conf(T)={}
NT=blue	Conf(NT)={WA}
Q=green	Conf(Q)={NSW, NT}
SA=?	Conf(SA)={WA, NSW, NT, Q}



Back-jumping: Example

Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?

WA=red Conf(WA)={}

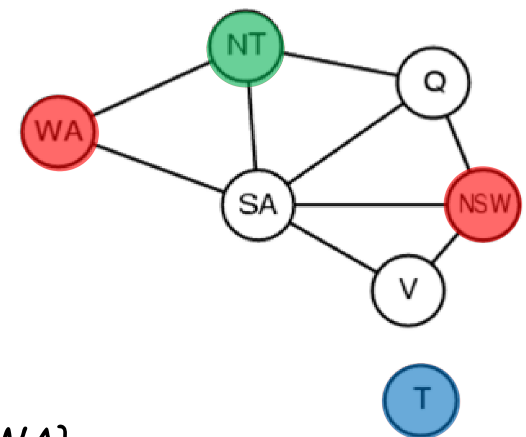
NSW=red Conf(NSW)={}

T=blue Conf(T)={}

NT=blue Conf(NT)={WA} NT=green Conf(NT)={WA}

Q=green Conf(Q)={NSW, NT}

SA=? Conf(SA)={WA, NSW, NT, Q}



Back-jumping: Example

Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?

WA=red Conf(WA)={}

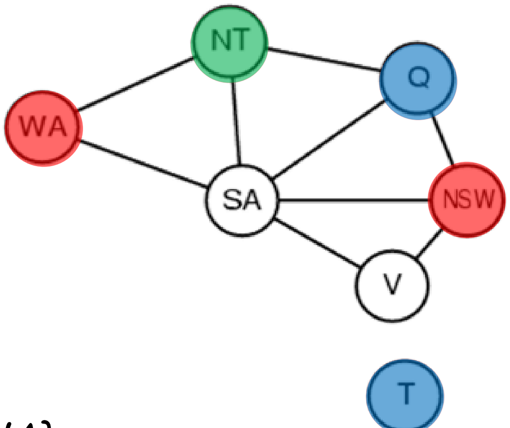
NSW=red Conf(NSW)={}

T=blue Conf(T)={}

NT=blue Conf(NT)={WA} NT=green Conf(NT)={WA}

Q=green Conf(Q)={NSW, NT} Q=blue Conf(Q)={NSW, NT}

SA=? Conf(SA)={WA, NSW, NT, Q}



Back-jumping: Example

Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?

WA=red — Conf(WA)={}

NSW=red — Conf(NSW)={}

T=blue — Conf(T)={}

NT=blue — Conf(NT)={WA}

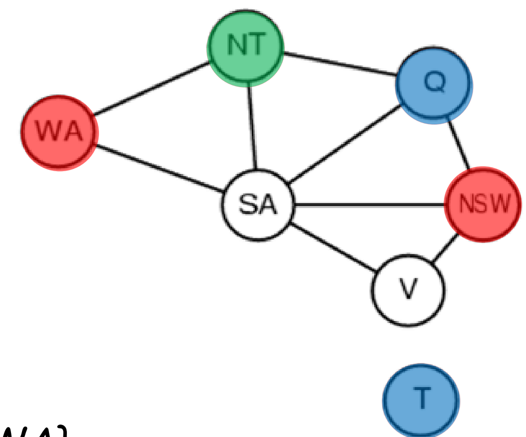
Q=green — Conf(Q)={NSW, NT}

SA=? — Conf(SA)={WA, NSW, NT, Q}

NT=green — Conf(NT)={WA}

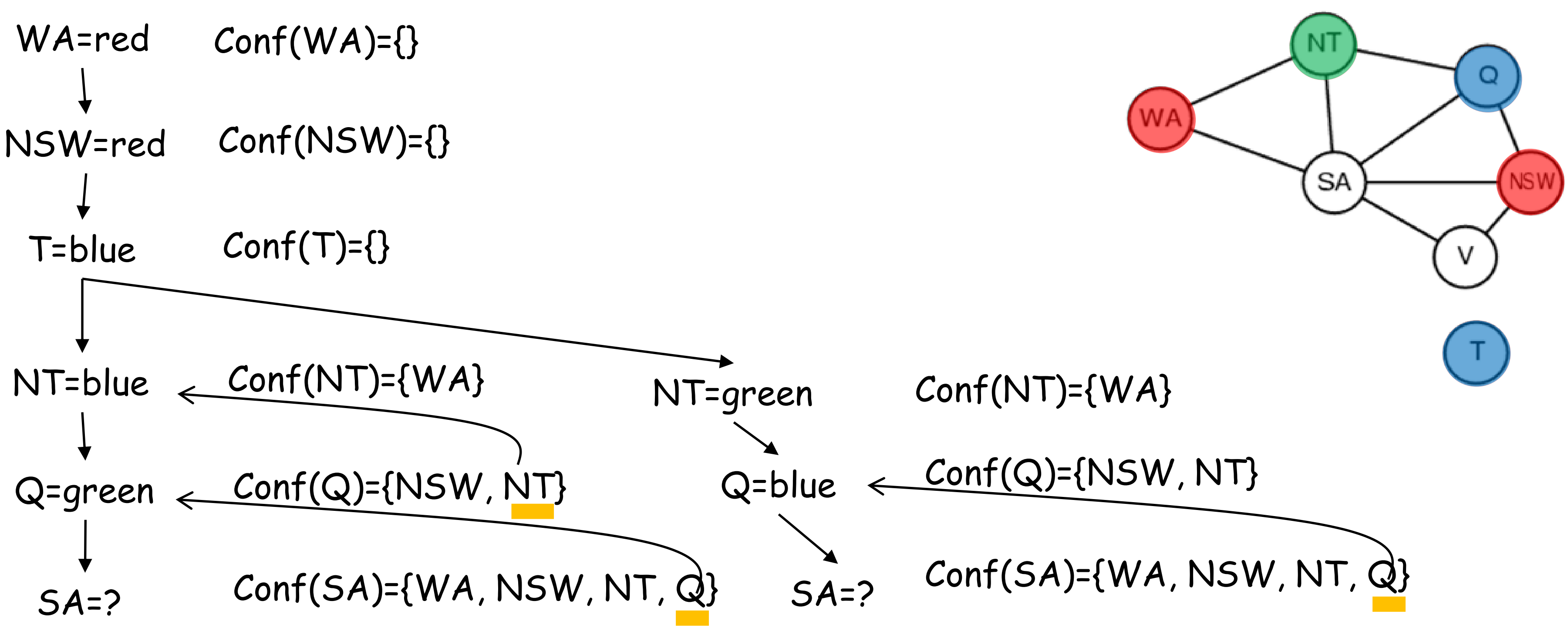
Q=blue — Conf(Q)={NSW, NT}

SA=? — Conf(SA)={WA, NSW, NT, Q}



Back-jumping: Example

Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?



Back-jumping: Example

Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?

WA=red Conf(WA)={}

NSW=red Conf(NSW)={}

T=blue Conf(T)={}

NT=blue Conf(NT)={WA}

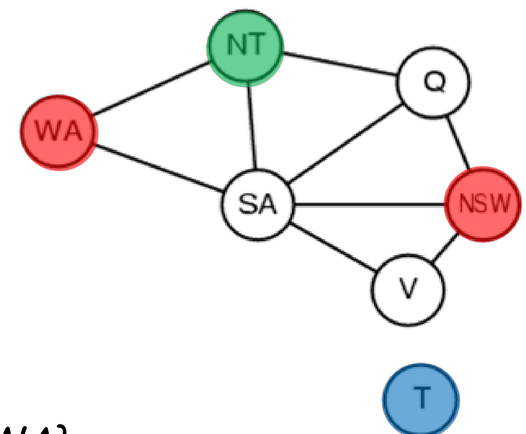
Q=green Conf(Q)={NSW, NT}

SA=? Conf(SA)={WA, NSW, NT, Q}

NT=green Conf(NT)={WA}

Q=blue Conf(Q)={NSW, NT}

SA=? Conf(SA)={WA, NSW, NT, Q}



Back-jumping: Example

Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?

WA=red   Conf(WA)={}

NSW=red   Conf(NSW)={}

T=blue   Conf(T)={}

NT=blue   Conf(NT)={WA}

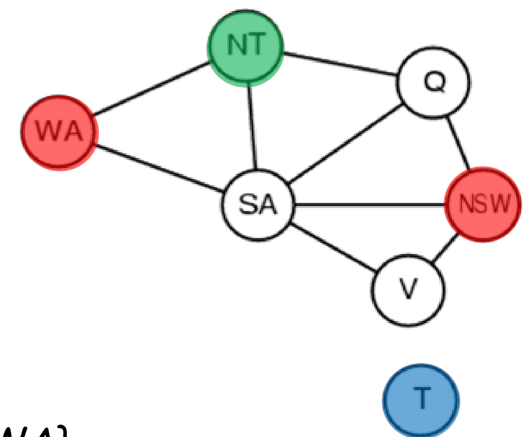
Q=green   Conf(Q)={NSW, NT}

SA=?   Conf(SA)={WA, NSW, NT, Q}

NT=green   Conf(NT)={WA}

Q=blue   Conf(Q)={NSW, NT}

SA=?   Conf(SA)={WA, NSW, NT, Q}



Back-jumping: Example

Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?

WA=red Conf(WA)={}

NSW=red Conf(NSW)={}

T=blue Conf(T)={}

NT=blue Conf(NT)={WA}

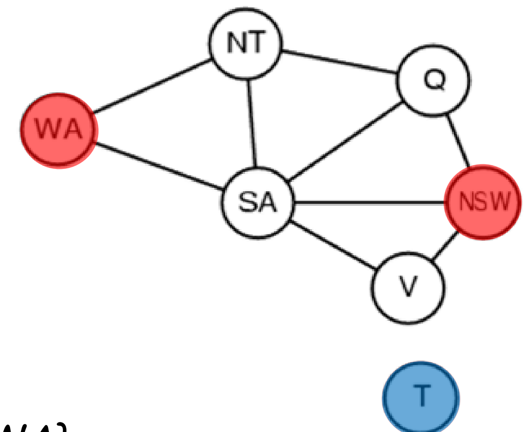
Q=green Conf(Q)={NSW, NT}

SA=? Conf(SA)={WA, NSW, NT, Q}

NT=green Conf(NT)={WA}

Q=blue Conf(Q)={NSW, NT}

SA=? Conf(SA)={WA, NSW, NT, Q}



Back-jumping: Example

Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?

WA=red　　Conf(WA)={}

NSW=red　　Conf(NSW)={}

T=blue　　Conf(T)={}

NT=blue　　Conf(NT)={WA}

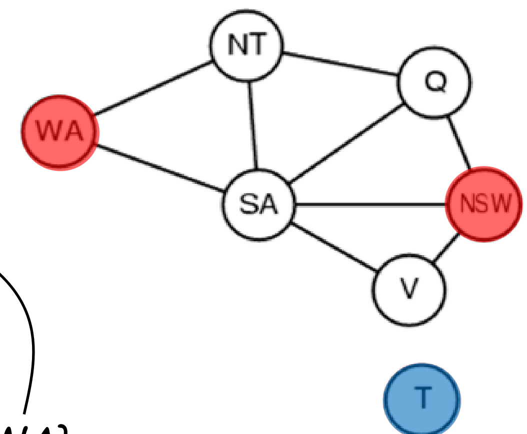
Q=green　　Conf(Q)={NSW, NT}

SA=?　　Conf(SA)={WA, NSW, NT, Q}

NT=green　　Conf(NT)={WA}

Q=blue　　Conf(Q)={NSW, NT}

SA=?　　Conf(SA)={WA, NSW, NT, Q}



Conflict-directed back-jumping

- Let X_j be the current variable, and let $conf(X_j)$ be its conflict set. If every possible value for X_j fails, back-jump to the most recent variable X_i in $conf(X_j)$, and set

$$conf(X_i) \leftarrow conf(X_i) \cup conf(X_j) - \{X_i\}$$

- When we reach a contradiction, back-jumping can tell us how far to back up, so we don't waste time changing variables that won't fix the problem.

Conflict-directed back-jumping: Example

Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?

WA=red	Conf(WA)={}

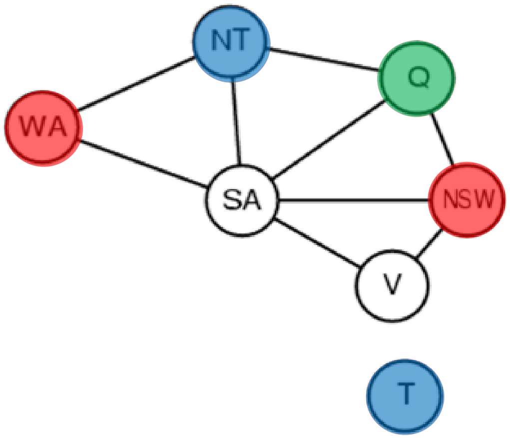
NSW=red	Conf(NSW)={}

T=blue	Conf(T)={}

NT=blue	Conf(NT)={WA}

Q=green	Conf(Q)={NSW, NT} → Conf(Q)={WA, NSW, NT}

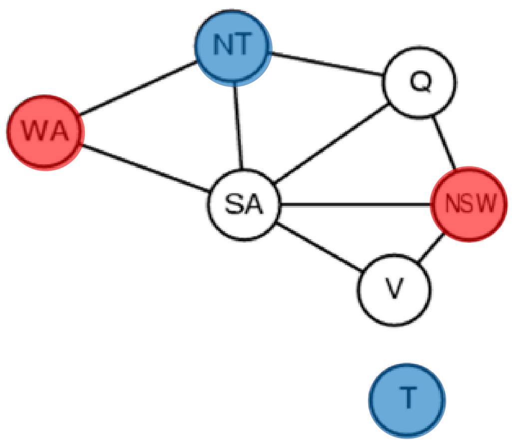
SA=?	Conf(SA)={WA, NSW, NT, Q}



The set of preceding variables that caused Q, together with any subsequent variables, have no consistent solution.

Conflict-directed back-jumping: Example

Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?



WA=red Conf(WA)={}

NSW=red Conf(NSW)={}

T=blue Conf(T)={}

NT=blue Conf(NT)={WA} → Conf(NT)={WA, NSW}

Q=green Conf(Q)={WA, NSW, NT}

SA=? Conf(SA)={WA, NSW, NT, Q}

The set of preceding variables that caused NT, together with any subsequent variables, have no consistent solution.

Conflict-directed back-jumping: Example

Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?

WA=red   Conf(WA)={}

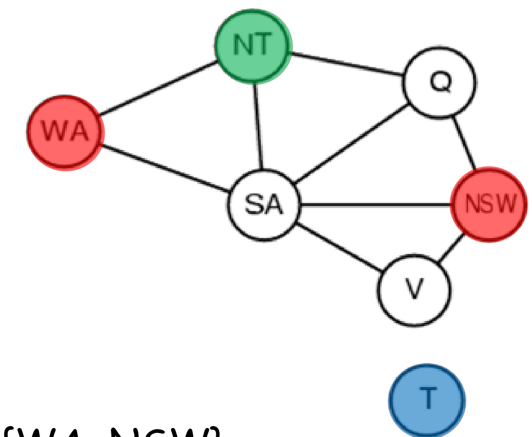
NSW=red   Conf(NSW)={}

T=blue   Conf(T)={}

NT=blue   Conf(NT)={WA, NSW}   NT=green   Conf(NT)={WA, NSW}

Q=green   Conf(Q)={WA, NSW, NT}

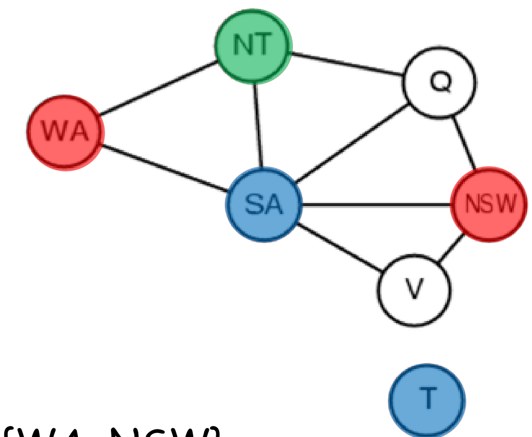
SA=?   Conf(SA)={WA, NSW, NT, Q}



Conflict-directed back-jumping: Example

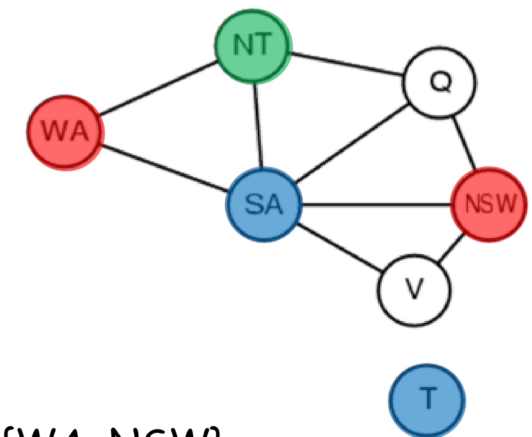
Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?

WA=red	Conf(WA)={}		
NSW=red	Conf(NSW)={}		
T=blue	Conf(T)={}		
NT=blue	Conf(NT)={WA, NSW}	NT=green	Conf(NT)={WA, NSW}
Q=green	Conf(Q)={WA, NSW, NT}	SA=blue	Conf(SA)={WA, NSW, NT}
SA=?	Conf(SA)={WA, NSW, NT, Q}		



Conflict-directed back-jumping: Example

Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?



WA=red	Conf(WA)={}		
NSW=red	Conf(NSW)={}		
T=blue	Conf(T)={}		
NT=blue	Conf(NT)={WA, NSW}	NT=green	Conf(NT)={WA, NSW}
Q=green	Conf(Q)={WA, NSW, NT}	SA=blue	Conf(SA)={WA, NSW, NT}
SA=?	Conf(SA)={WA, NSW, NT, Q}	Q=?	Conf(Q)={NSW, NT, SA}

Conflict-directed back-jumping: Example

Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?

WA=red Conf(WA)={}

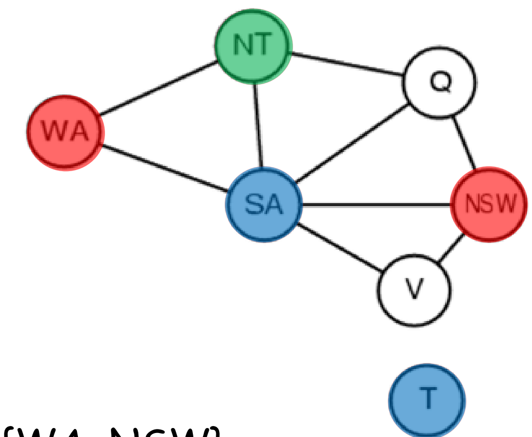
NSW=red Conf(NSW)={}

T=blue Conf(T)={}

NT=blue Conf(NT)={WA, NSW} NT=green Conf(NT)={WA, NSW}

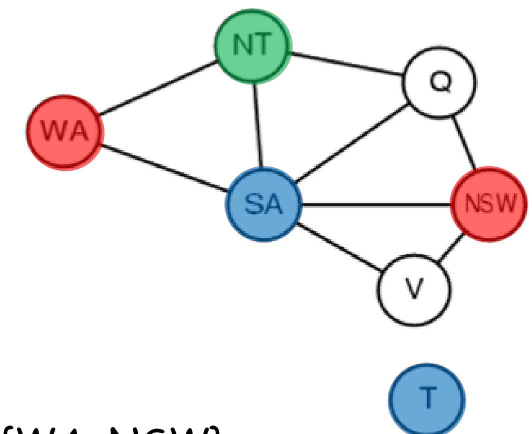
Q=green Conf(Q)={WA, NSW, NT} SA=blue Conf(SA)={WA, NSW, NT}

SA=? Conf(SA)={WA, NSW, NT, Q} Q=? Conf(Q)={NSW, NT, SA}



Conflict-directed back-jumping: Example

Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?



WA=red Conf(WA)={}

NSW=red Conf(NSW)={}

T=blue Conf(T)={}

NT=blue Conf(NT)={WA, NSW}

Q=green Conf(Q)={WA, NSW, NT}

SA=? Conf(SA)={WA, NSW, NT, Q}

NT=green Conf(NT)={WA, NSW}

SA=blue Conf(SA)={WA, NSW, NT}

Q=? Conf(Q)={NSW, NT, SA}

Conflict-directed back-jumping: Example

Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?

WA=red Conf(WA)={}

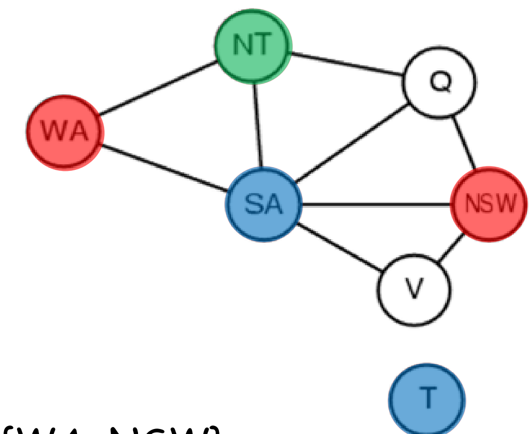
NSW=red Conf(NSW)={}

T=blue Conf(T)={}

NT=blue Conf(NT)={WA, NSW} NT=green Conf(NT)={WA, NSW}

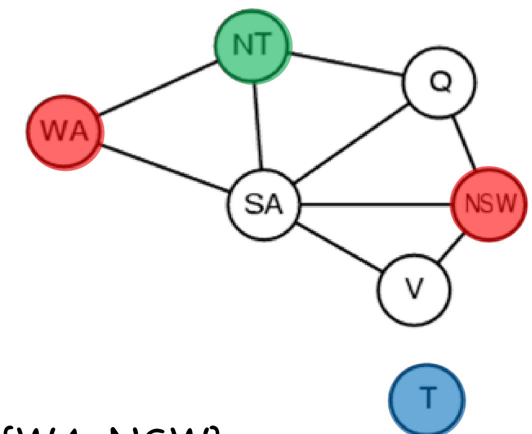
Q=green Conf(Q)={WA, NSW, NT} SA=blue Conf(SA)={WA, NSW, NT}

SA=? Conf(SA)={WA, NSW, NT, Q} Q=? Conf(Q)={NSW, NT, SA}



Conflict-directed back-jumping: Example

Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?



WA=red Conf(WA)={}

NSW=red Conf(NSW)={}

T=blue Conf(T)={}

NT=blue Conf(NT)={WA, NSW}

Q=green Conf(Q)={WA, NSW, NT}

SA=? Conf(SA)={WA, NSW, NT, Q}

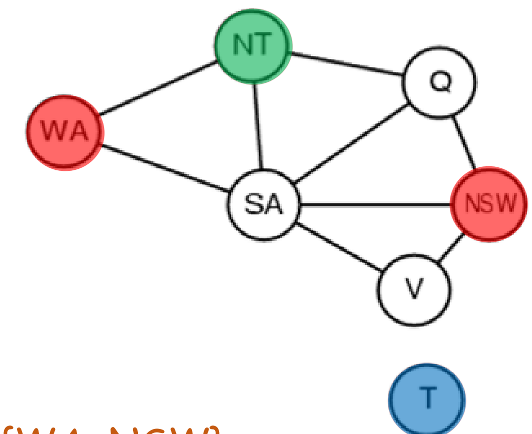
NT=green Conf(NT)={WA, NSW}

SA=blue Conf(SA)={WA, NSW, NT}

Q=? Conf(Q)={NSW, NT, SA}

Conflict-directed back-jumping: Example

Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?



WA=red   Conf(WA)={}

NSW=red   Conf(NSW)={}

T=blue   Conf(T)={}

NT=blue   Conf(NT)={WA, NSW}   NT=green   Conf(NT)={WA, NSW}

Q=green   Conf(Q)={WA, NSW, NT}   SA=blue   Conf(SA)={WA, NSW, NT}

SA=?   Conf(SA)={WA, NSW, NT, Q}   Q=?   Conf(Q)={NSW, NT, SA}

Conflict-directed back-jumping: Example

Suppose we have generated the partial assignment
WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?

WA=red Conf(WA)={}

NSW=red Conf(NSW)={} → Conf(NSW)={WA}

T=blue Conf(T)={}

NT=blue Conf(NT)={WA, NSW}

Q=green Conf(Q)={WA, NSW, NT}

SA=? Conf(SA)={WA, NSW, NT, Q}

NT=green Conf(NT)={WA, NSW}

SA=blue Conf(SA)={WA, NSW, NT}

Q=? Conf(Q)={NSW, NT, SA}