CONSTRAINT SATISFACTION PROBLEMS

Constraint satisfaction problems (CSPs)

Standard search problem:

state is a "black box"—any old data structure
that supports goal test, eval, successor

CSP:

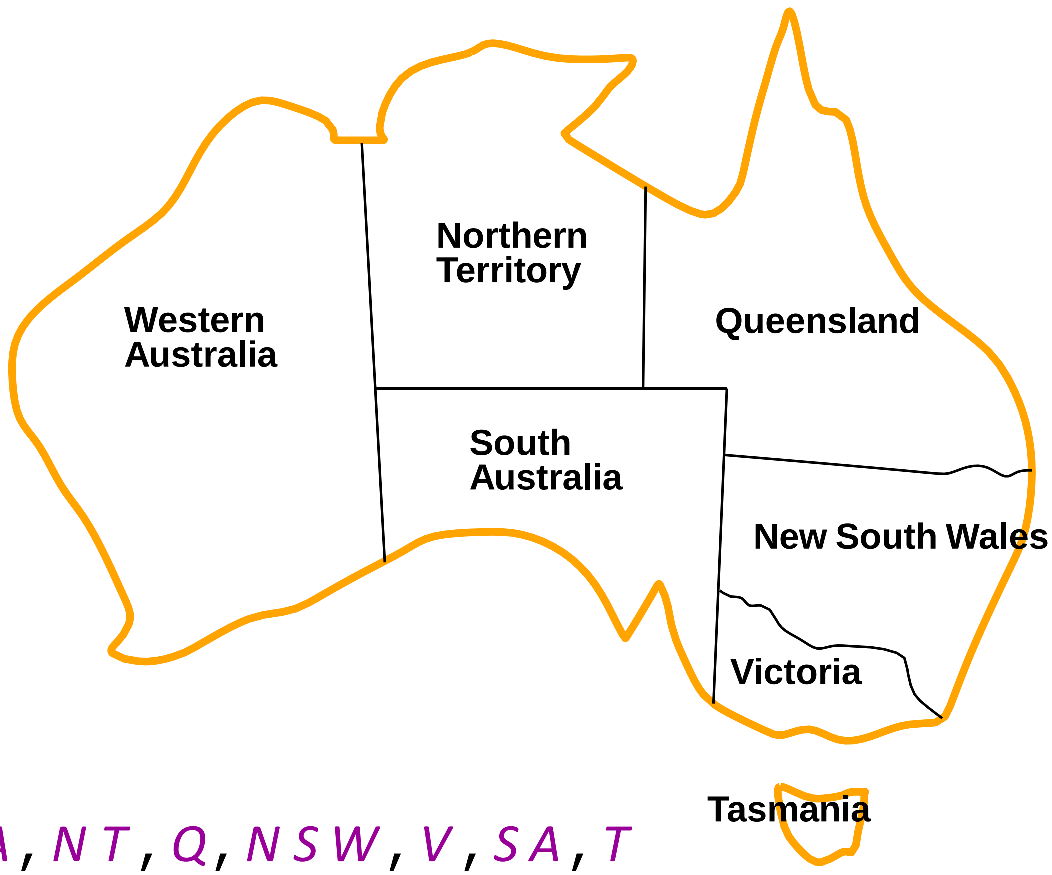
state is defined by variables X_i with values from domain D_i

goal test is a set of constraints specifying
allowable combinations of values for subsets of variables

Simple example of a formal representation language

Allows useful general-purpose algorithms with more power than standard search algorithms

Example: Map-Coloring



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

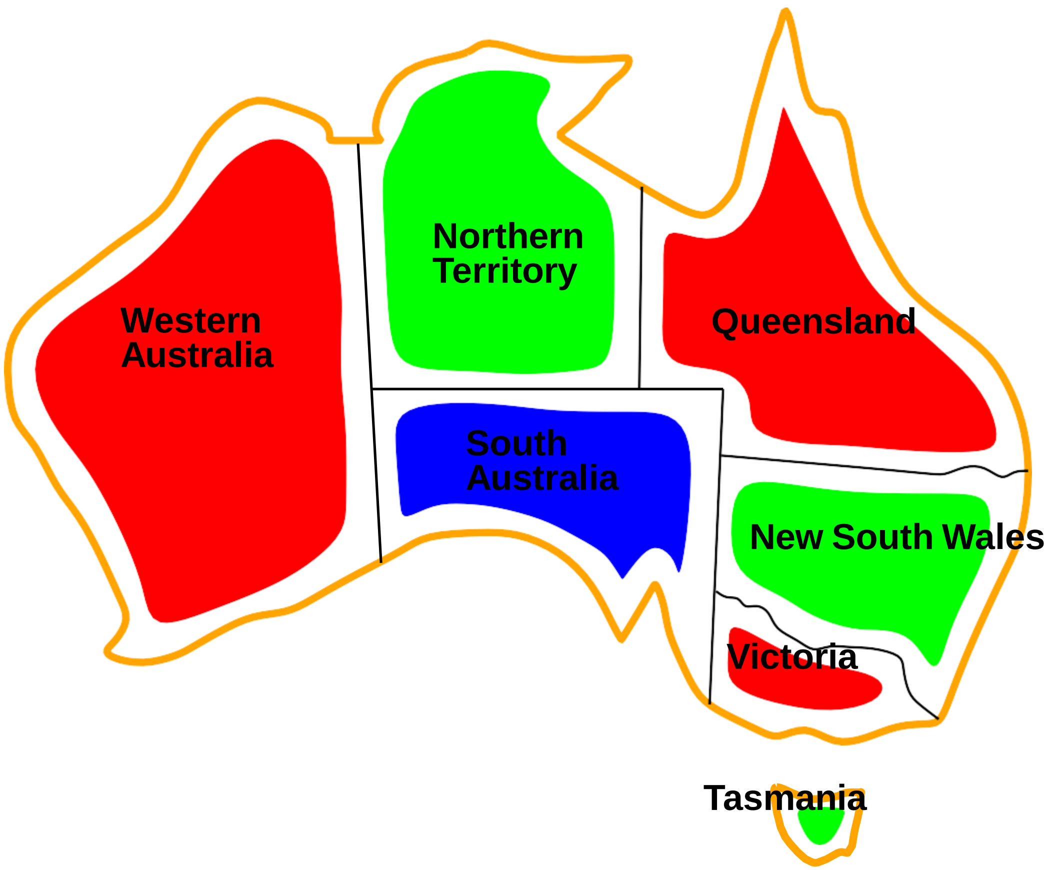
Domains $D_i = \{red, green, blue\}$

Constraints: adjacent regions must have different colors

e.g., $WA \neq NT$ (if the language allows this), or

$(WA, NT) \in \{(red, green), (red, blue), (green, red), (green, blue), \ldots\}$

Example: Map-Coloring contd.

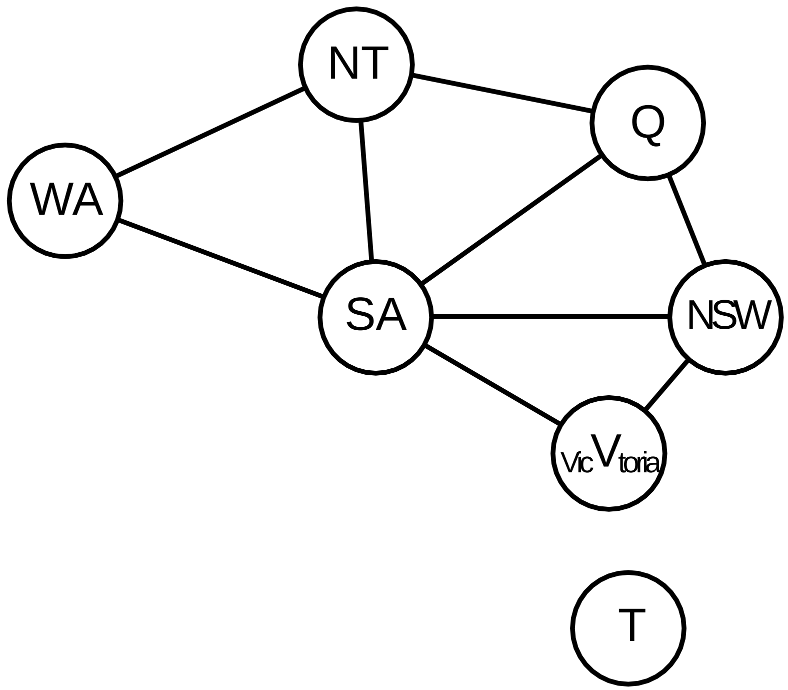


Solutions are assignments satisfying all constraints, e.g.,
$\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green\}$

Constraint graph

Binary CSP: each constraint relates at most two variables

Constraint graph: nodes are variables, arcs show constraints



General-purpose CSP algorithms use the graph structure
to speed up search. E.g., Tasmania is an independent subproblem!

Varieties of CSPs

Discrete variables

finite domains; size $d \Rightarrow O(d^n)$ complete assignments

- e.g., Boolean CSPs, incl. Boolean satisfiability (NP-complete)

infinite domains (integers, strings, etc.)

- e.g., job scheduling, variables are start/end days for each job
- need a constraint language, e.g., $StartJob_1 + 5 \leq StartJob_3$
- linear constraints solvable, nonlinear undecidable

Continuous variables

- e.g., start/end times for Hubble Telescope observations
- linear constraints solvable in poly time by LP methods

Varieties of constraints

Unary constraints involve a single variable,
e.g., $SA \neq green$

Binary constraints involve pairs of variables,
e.g., $SA \neq WA$

Higher-order constraints involve 3 or more variables,
e.g., cryptarithmetic column constraints

Preferences (soft constraints), e.g., red is better than $green$
often representable by a cost for each variable assignment
→ constrained optimization problems

Real-world CSPs

Assignment problems
- e.g., who teaches what class

Timetabling problems
- e.g., which class is offered when and where?

Hardware configuration

Spreadsheets

Transportation scheduling

Factory scheduling

Floorplanning

Notice that many real-world problems involve real-valued variables

Standard search formulation (incremental)

Let's start with the straightforward, dumb approach, then fix it

States are defined by the values assigned so far

- Initial state: the empty assignment, { }
- Successor function: assign a value to an unassigned variable that does not conflict with current assignment.
  $\Rightarrow$ fail if no legal assignments (not fixable!)
- Goal test: the current assignment is complete

1) This is the same for all CSPs!
2) Every solution appears at depth n with n variables
   $\Rightarrow$ use depth-first search
3) Path is irrelevant, so can also use complete-state formulation
4) $b = (n - l)d$ at depth l, hence $n!d^n$ leaves!!!!

Backtracking search

Variable assignments are commutative, i.e.,

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

Only need to consider assignments to a single variable at each node

$\Rightarrow$ $b = d$ and there are d^n leaves

Depth-first search for CSPs with single-variable assignments is called backtracking search

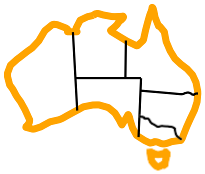
Backtracking search is the basic uninformed algorithm for CSPs

Can solve n-queens for $n \approx 25$

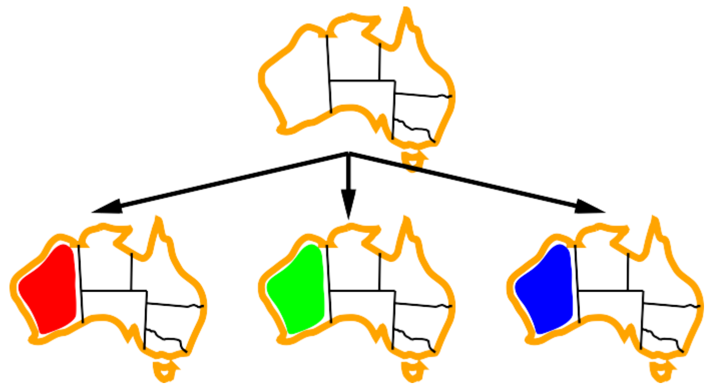
Backtracking search

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
   return RECURSIVE-BACKTRACKING({ },csp)
function RECURSIVE-BACKTRACKING(assignment,csp) returns soln/failure
   if assignment is complete then return assignment
   var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp],assignment,csp)
   for each value in ORDER-DOMAIN-VALUES(var,assignment,csp) do
      if value is consistent with assignment given CONSTRAINTS[csp] then
         add {var = value} to assignment
         result ← RECURSIVE-BACKTRACKING(assignment,csp)
         if result /= failure then return result
         remove {var = value} from assignment
   return failure
```

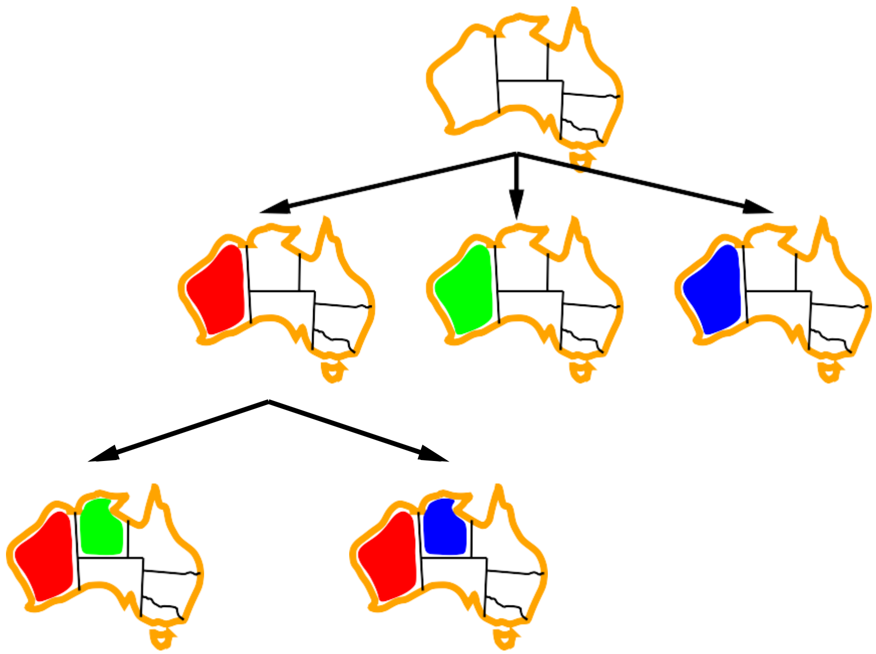
Backtracking example



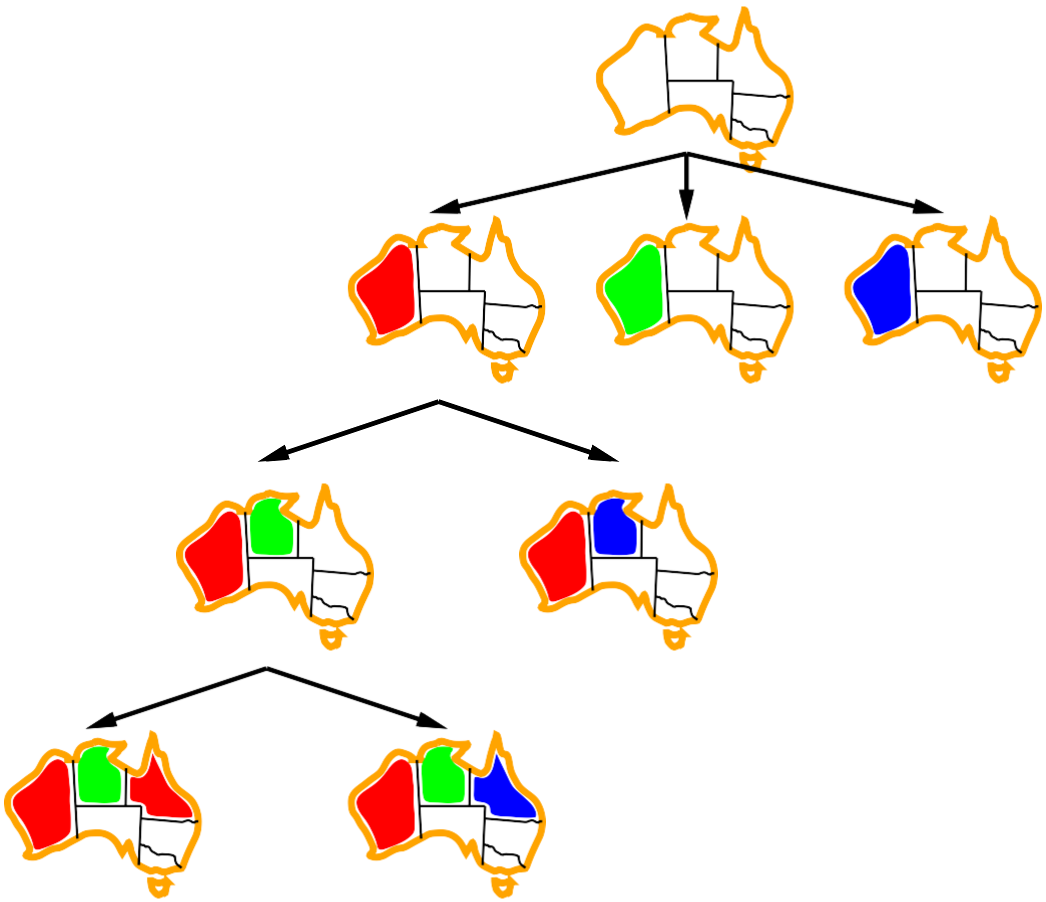
Backtracking example



Backtracking example



Backtracking example



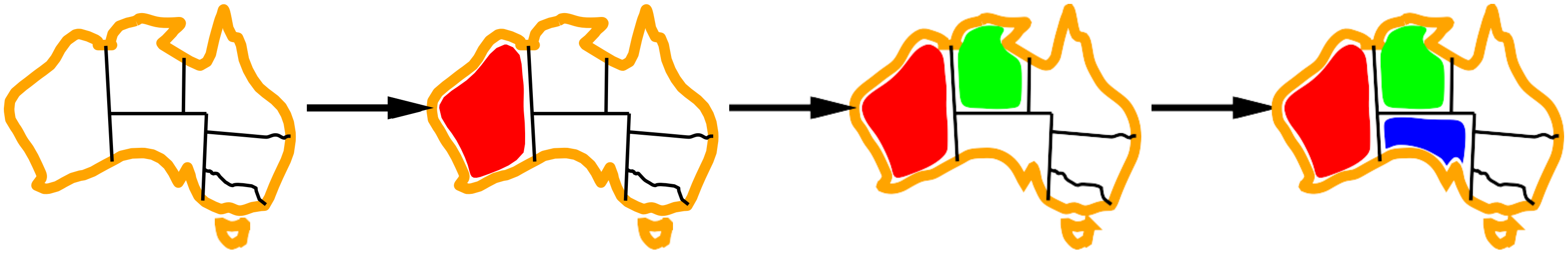
Improving backtracking efficiency

General-purpose methods can give huge gains in speed:

1. Which variable should be assigned next?
2. In what order should its values be tried?
3. Can we detect inevitable failure early?
4. Can we take advantage of problem structure?

Minimum remaining values

Minimum remaining values (MRV):
choose the variable with the fewest legal values

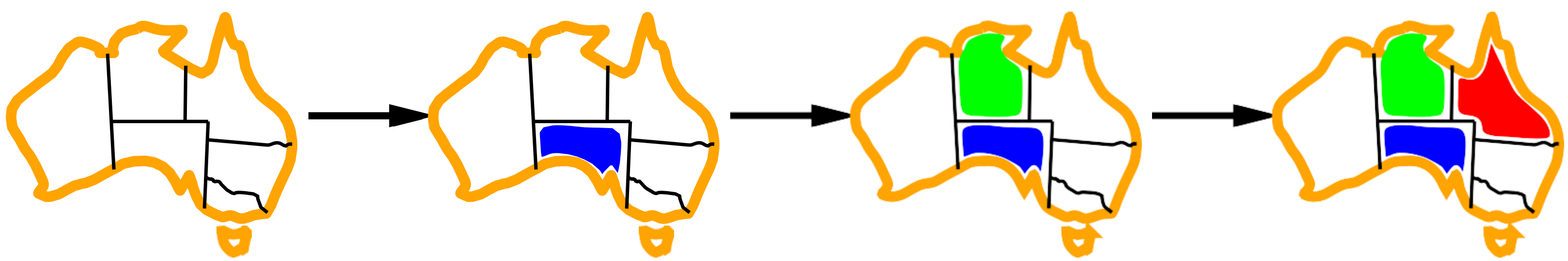


Degree heuristic

Tie-breaker among MRV variables

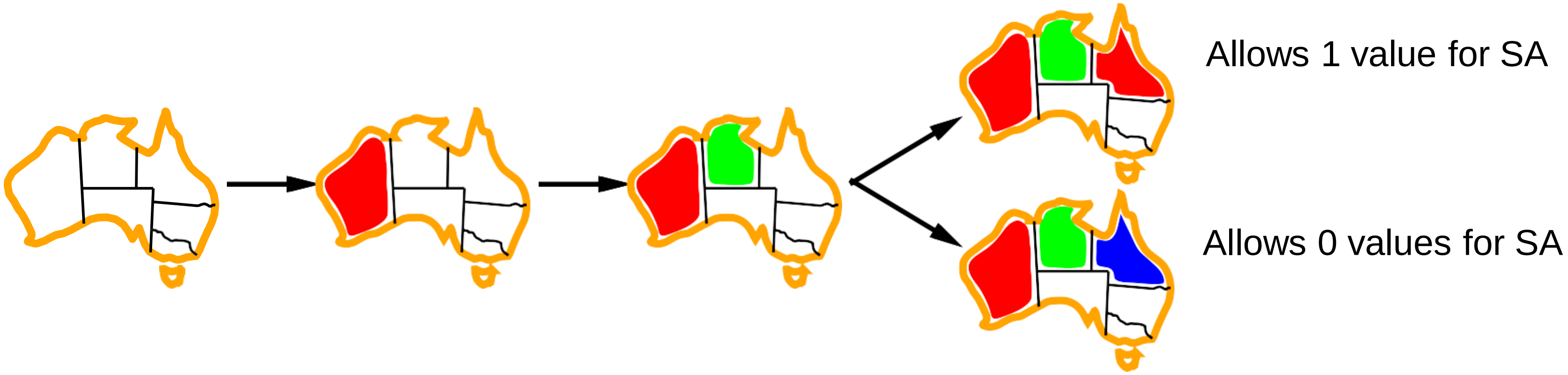
Degree heuristic:

choose the variable with the most constraints on remaining variables



Least constraining value

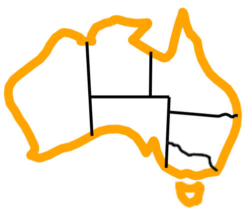
Given a variable, choose the least constraining value:
the one that rules out the fewest values in the remaining variables



Combining these heuristics makes 1000 queens feasible

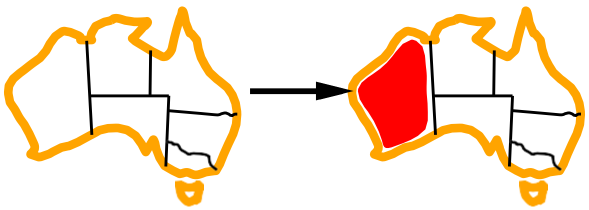
Forward checking

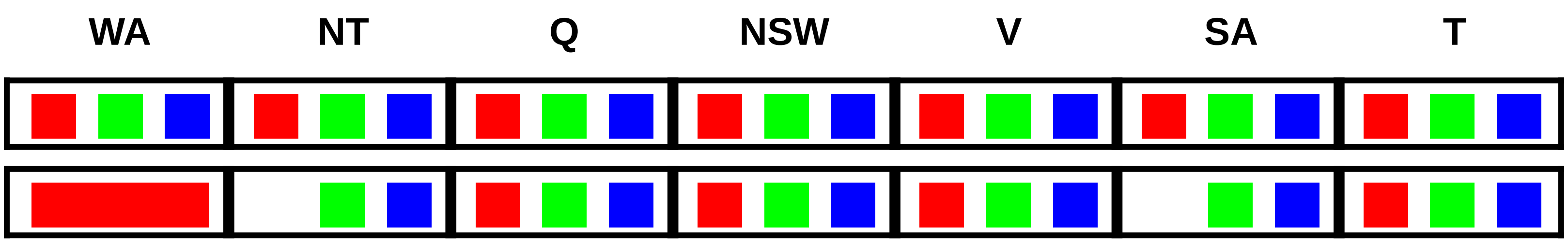
Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values



Forward checking

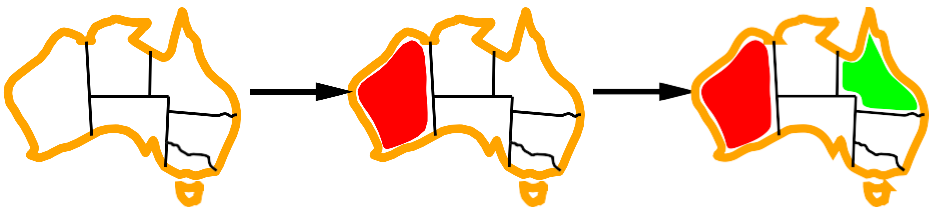
Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values

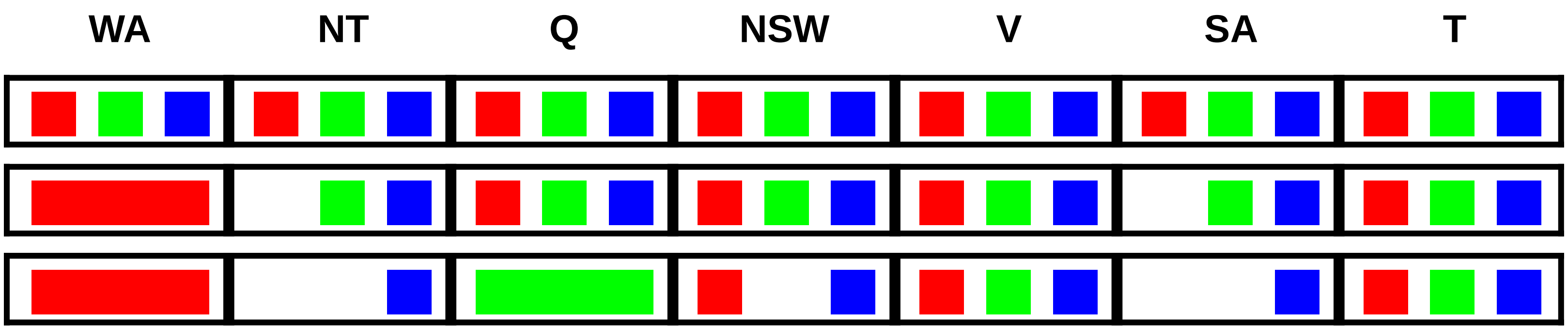




Forward checking

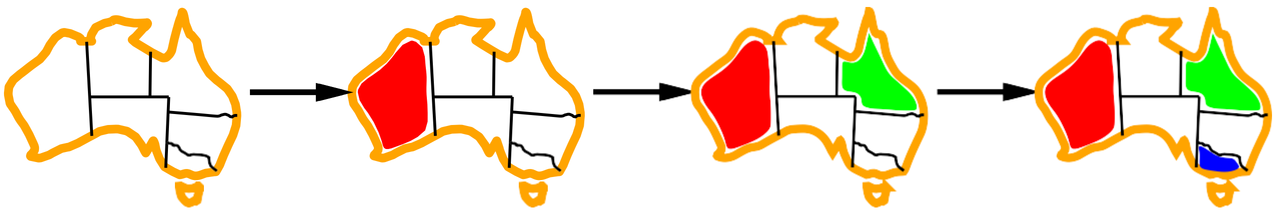
Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values

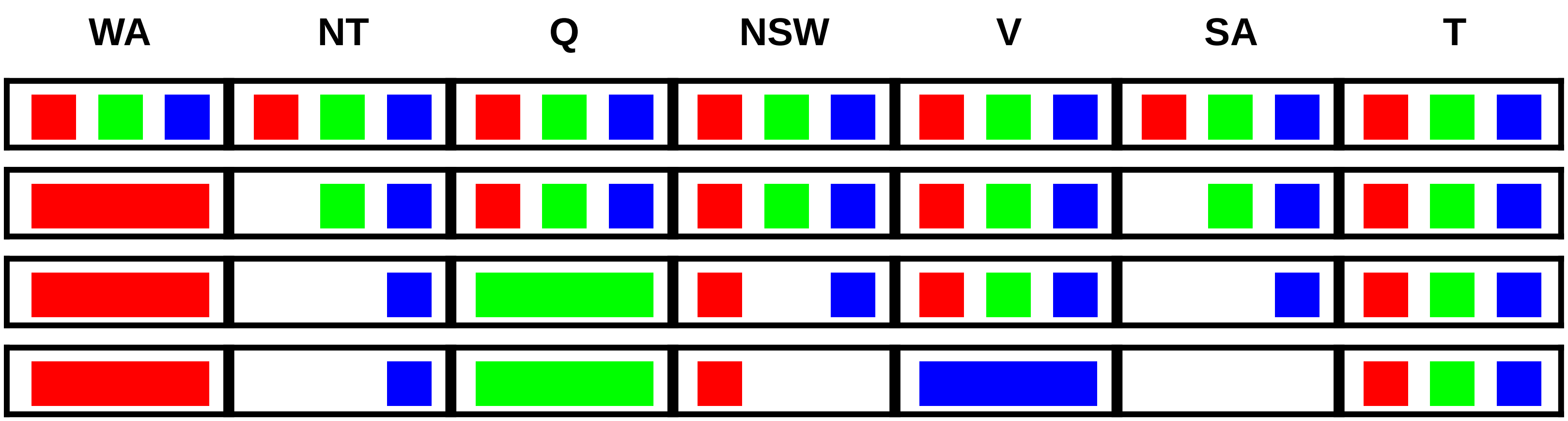




Forward checking

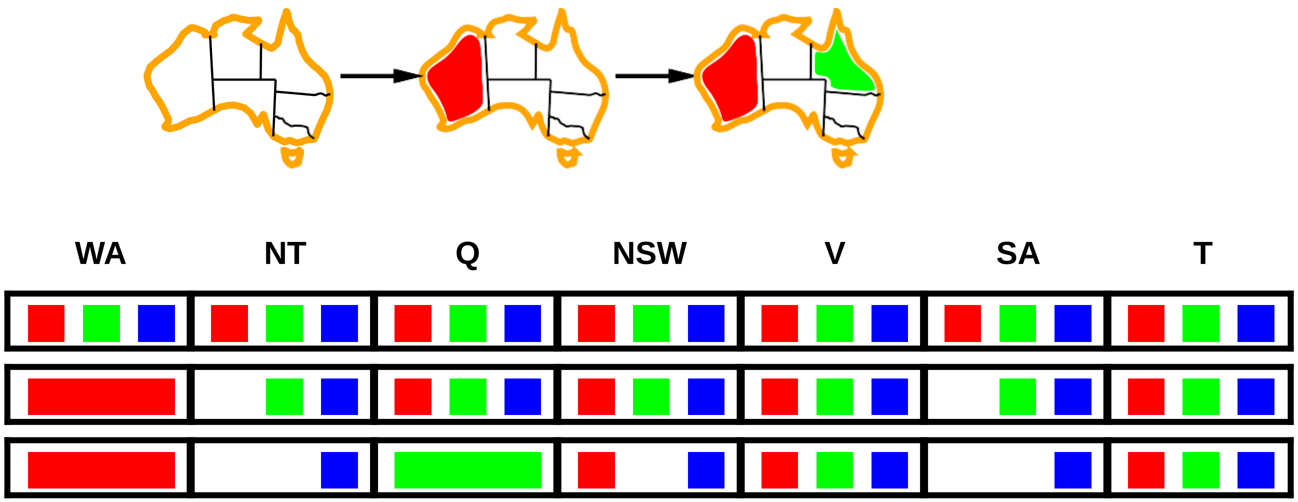
Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values





Constraint propagation

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



NT and SA cannot both be blue!

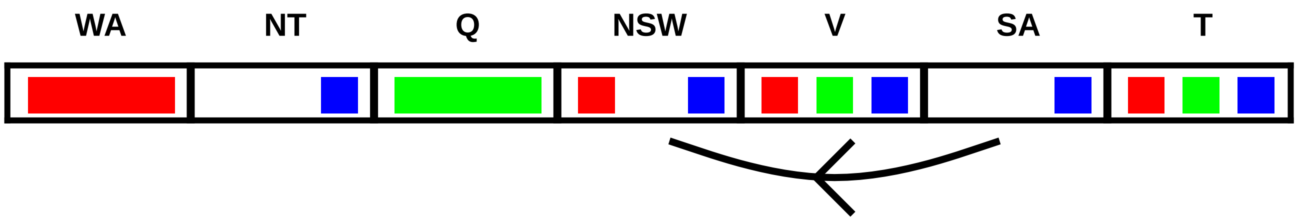
Constraint propagation repeatedly enforces constraints locally

Arc consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff

for every value x of X there is some allowed y

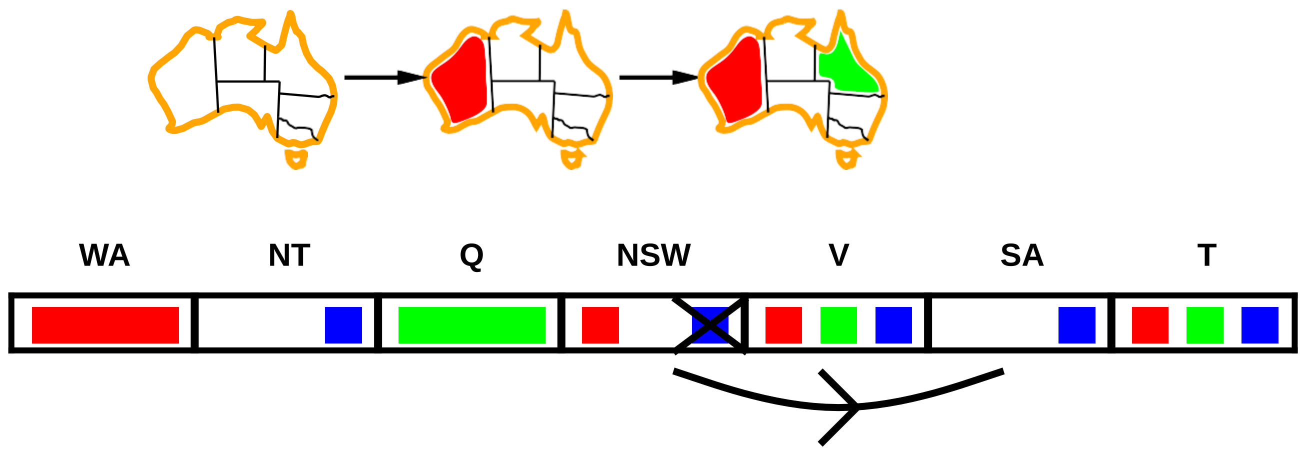


Arc consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff

for every value x of X there is some allowed y

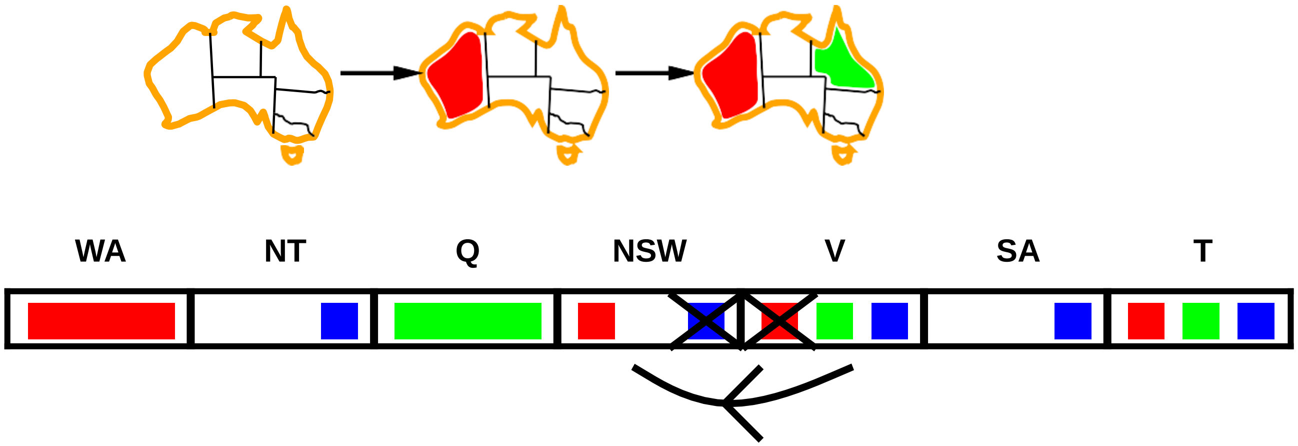


Arc consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff

for every value x of X there is some allowed y



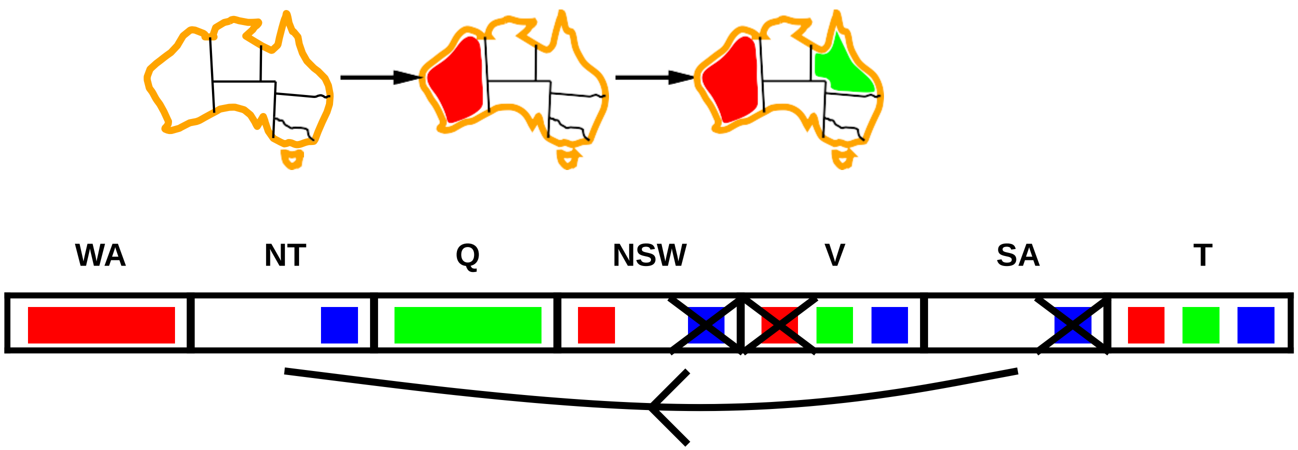
If X loses a value, neighbors of X need to be rechecked

Arc consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff

for every value x of X there is some allowed y



If X loses a value, neighbors of X need to be rechecked

Arc consistency detects failure earlier than forward checking

Can be run as a preprocessor or after each assignment

Arc consistency algorithm

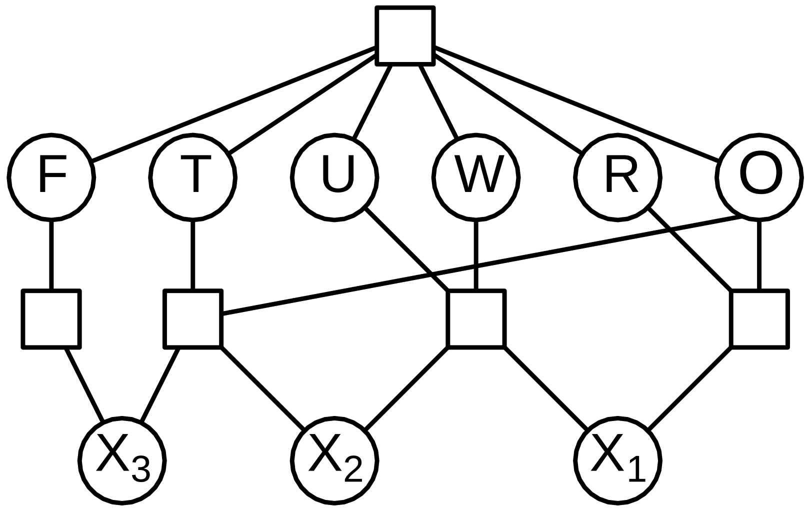
```
function AC-3(csp) returns the CSP, possibly with reduced domains
   inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
   local variables: queue, a queue of arcs, initially all the arcs in csp

   while queue is not empty do
      (Xi, Xj) ← REMOVE-FIRST(queue)
      if REMOVE-INCONSISTENT-VALUES(Xi, Xj) then
         for each Xk in NEIGHBORS[Xi] do
            add (Xk, Xi) to queue
```

```
function REMOVE-INCONSISTENT-VALUES(Xi, Xj) returns true iff succeeds
   removed ← false
   for each x in DOMAIN[Xi] do
      if no value y in DOMAIN[Xj] allows (x,y) to satisfy the constraint Xi ↔ Xj
         then delete x from DOMAIN[Xi]; removed ← true
   return removed
```

$O(n^2d^3)$, can be reduced to $O(n^2d^2)$ (but detecting all is NP-hard)

Cryptarithmetic

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



Variables: $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$

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

Constraints

alldiff(F, T, U, W, R, O)

$O + O = R + 10 \cdot X_1$, etc.

Crypt-Arithmetic puzzle

Problem Statement:

- Solve the following puzzle by assigning numeral (0-9) in such a way that each letter is assigned unique digit which satisfy the following addition.
- Constraints : No two letters have the same value. (The constraints of arithmetic).

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

Initial Problem State

S = ? ; E = ? ;N = ? ; D = ? ; M = ? ;O = ? ; R = ? ;Y = ?

Carries : $C_4 = ?$; $C_3 = ?$; $C_2 = ?$; $C_1 = ?$

$\mathbf{C_4}$	$\mathbf{C_3}$	$\mathbf{C_2}$	$\mathbf{C_1}$	$\leftarrow$	**Carry**
	S	**E**	**N**	**D**	
+	**M**	**O**	**R**	**E**	
M	**O**	**N**	**E**	**Y**	

Constraint equations:

$$Y = D + E$$

$$E = N + R + C_1$$

$$N = E + O + C_2$$

$$O = S + M + C_3$$

$$M = C_4$$

We can easily see that M has to be non zero digit, so the value of C4 =1

1. M = C4 $\Rightarrow$ $\boxed{\mathbf{M = 1}}$
2. O = S + M + C3 → C4

   For C4 =1, S + M + C3 > 9 $\Rightarrow$

   S + 1 + C3 > 9 $\Rightarrow$ S+C3 > 8.

   If C3 = 0, then S = 9 else if C3 = 1,

   then S = 8 or 9.

We see that for S = 9

C3 = 0 or 1

It can be easily seen that C3 = 1 is not possible as O = S + M + C3 $\Rightarrow$ O = 11 $\Rightarrow$ O has to be assigned digit 1 but 1 is already assigned to M, so not possible.

Therefore, only choice for C3 = 0, and thus O = 10. This implies that O is assigned 0 (zero) digit.

Therefore, O = 0

$\boxed{\mathbf{M = 1, O = 0}}$

C_4	C_3	C_2	C_1	←	Carry
	S	E	N	D	
+	M	O	R	E	
M	O	N	E	Y	

Y = D + E

E = N + R + C1

N = E + O + C2

O = S + M + C3

M = C4

3. Since C3 = 0; N = E + O + C2 produces no carry.

As O = 0, N = E + C2 .

Since N $\neq$ E, therefore, C2 = 1.

Hence N = E + 1

Now E can take value from 2 to 8 {0,1,9 already assigned so far }

- If E = 2, then N = 3.
- Since C2 = 1, from E = N + R + C1 , we get
  - 12 = N + R + C1
    - If C1 = 0 then R = 9, which is not possible as we are on the path with S = 9
    - If C1 = 1 then R = 8, then
      - From Y = D + E , we get 10 + Y= D + 2 .
      - For no value of D, we can get Y.
- Try similarly for E = 3, 4. We fail in each case.

C_4	C_3	C_2	C_1	←	Carry
	S	E	N	D	
+	M	O	R	E	
M	O	N	E	Y	

Y = D + E

E = N + R + C1

N = E + O + C2

O = S + M + C3

M = C4

If E = 5, then N = 6

Since C2 = 1, from E = N + R + C1 , we get 15 = N + R + C1 ,

If C1 = 0 then R = 9, which is not possible as we are on the path with S = 9.

If C1 = 1 then R = 8, then

From Y = D + E , we get 10 + Y= D + 5 i.e., 5 + Y = D.

If Y = 2 then D = 7. These values are possible.

Hence we get the final solution as given below and on backtracking, we may find more solutions.

S = 9 ; E = 5 ; N = 6 ; D = 7 ;
M = 1 ; O = 0 ; R = 8 ;Y = 2

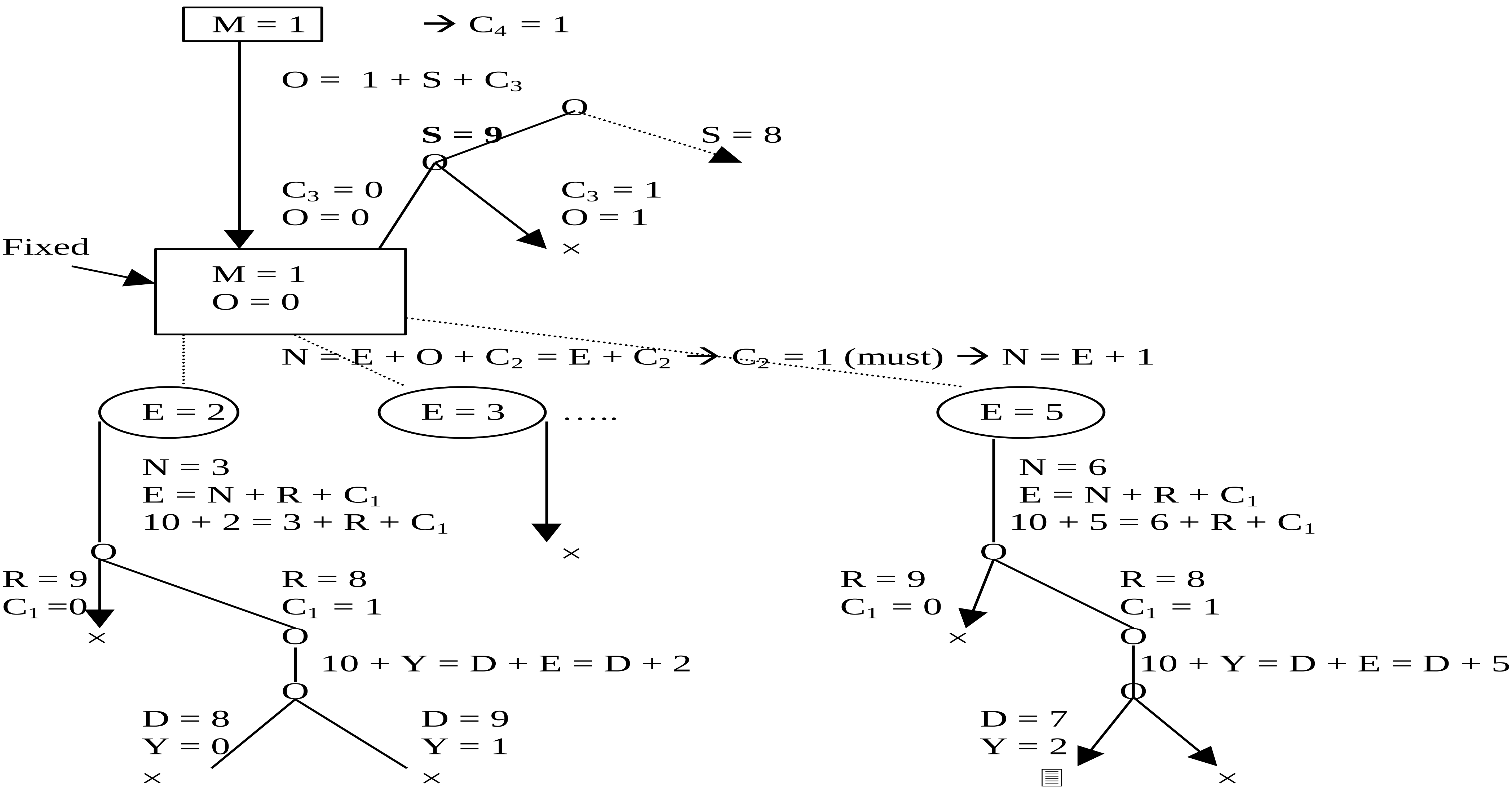
C_4	C_3	C_2	C_1	←	Carry
	S	E	N	D	
+	M	O	R	E	
M	O	N	E	Y	

Y = D + E
E = N + R + C1
N = E + O + C2
O = S + M + C3
M = C4

Constraints:

$Y = D + E \longrightarrow C_1$
$E = N + R + C_1 \longrightarrow C_2$
$N = E + O + C_2 \longrightarrow C_3$
$O = S + M + C_3 \longrightarrow C_4$
$M = C_4$

Initial State



The first solution obtained is:

M = 1, O = 0, S = 9, E = 5, N = 6, R = 8, D = 7, Y = 2

C4	C3	C2	C1		← Carries
	B	A	S	E	
+	B	A	L	L	
G	A	M	E	S	

Constraints equations are:

$E + L = S \rightarrow C1$

$S + L + C1 = E \rightarrow C2$

$2A + C2 = M \rightarrow C3$

$2B + C3 = A \rightarrow C4$

$G = C4$

Initial Problem State

G = ?; A = ?;M = ?; E = ?; S = ?; B = ?; L = ?

1. $G = C_4 \Rightarrow$ **G = 1**

2. $2B + C_3 = A \quad \rightarrow C_4$

2.1 Since $C_4 = 1$, therefore, $2B + C_3 > 9 \Rightarrow$ B can take values from 5 to 9.

2.2 Try the following steps for each value of B from 5 to 9 till we get a possible value of B.

if $C_3 = 0 \Rightarrow A = 0 \Rightarrow M = 0$ for $C_2 = 0$ or $M = 1$ for $C_2 = 1$ ×

- If B = 5

if $C_3 = 1 \Rightarrow A = 1$ × (as G = 1 already)

- For B = 6 we get similar contradiction while generating the search tree.
- If $\boxed{\mathbf{B = 7}}$, then for $C_3 = 0$, we get $\boxed{\mathbf{A = 4}}$ $\Rightarrow M = 8$ if $C_2 = 0$ that leads to contradiction, so this path is pruned. If $C_2 = 1$, then $\boxed{\mathbf{M = 9}}$.

3. Let us solve $S + L + C_1 = E$ and $E + L = S$

- Using both equations, we get $2L + C_1 = 0 \Rightarrow$ $\boxed{\mathbf{L = 5}}$ and $C_1 = 0$
- Using L = 5, we get $S + 5 = E$ that should generate carry $C_2 = 1$ as shown above
- So $S+5 > 9 \Rightarrow$ Possible values for E are {2, 3, 6, 8} (with carry bit $C_2 = 1$)
- If E = 2 then $S + 5 = 12 \Rightarrow S = 7$ × (as B = 7 already)
- If E = 3 then $S + 5 = 13 \Rightarrow S = 8$.
- Therefore $\boxed{\mathbf{E = 3}}$ and $\boxed{\mathbf{S = 8}}$ are fixed up.

4. Hence we get the final solution as given below and on backtracking, we may find more solutions. In this case we get only one solution.

G = 1; A = 4; M = 9; E = 3; S = 8;B = 7; L = 5

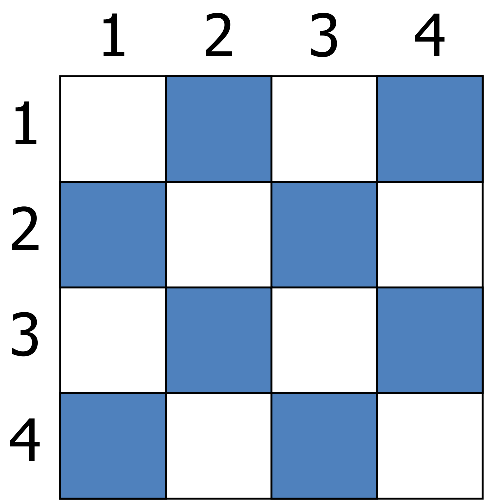
4-Queen

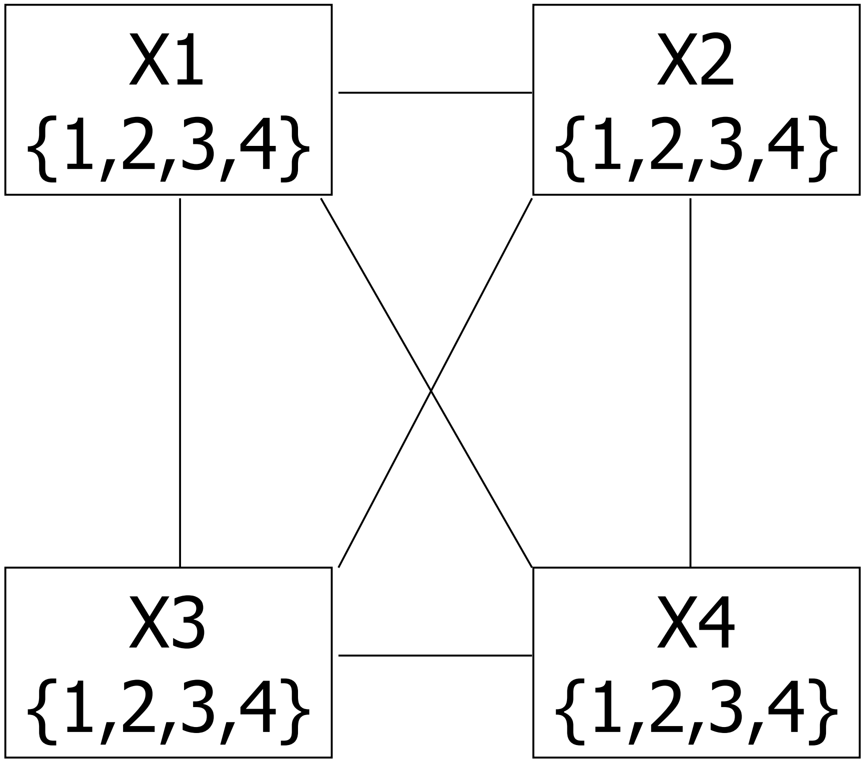
Variables: x_1, x_2, x_3, x_4 where x_i is the row position of the queen in column i, where i ∈ {0, 1, 2, 3}.

Domains: {0, 1, 2, 3}

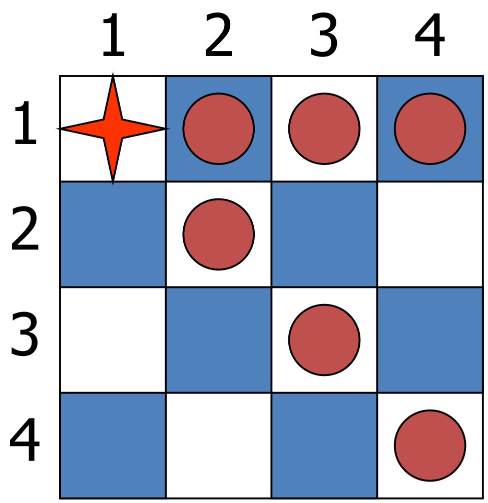
Constraints :
(a) Column Constarint : $i \neq j$
(b) Row Constraints : $x_i \neq x_j$
(c) Diagonal Constraints : $\left|x_i - x_j\right| \neq |i - j|$

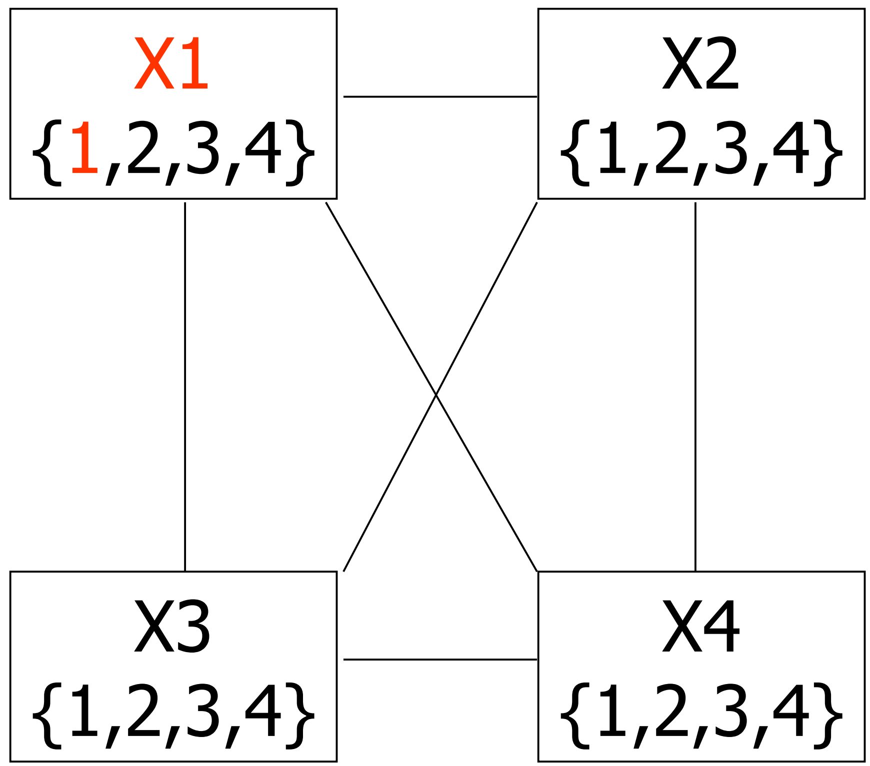
4-Queens Problem



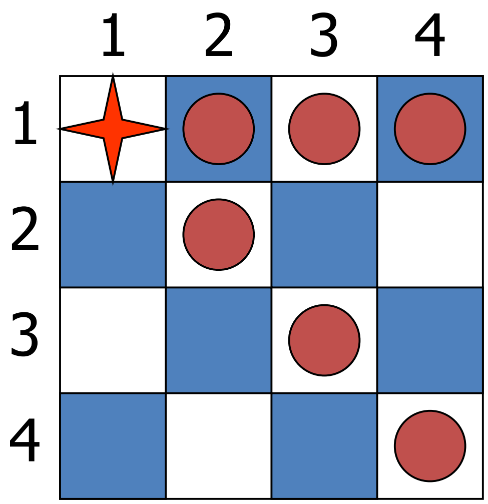


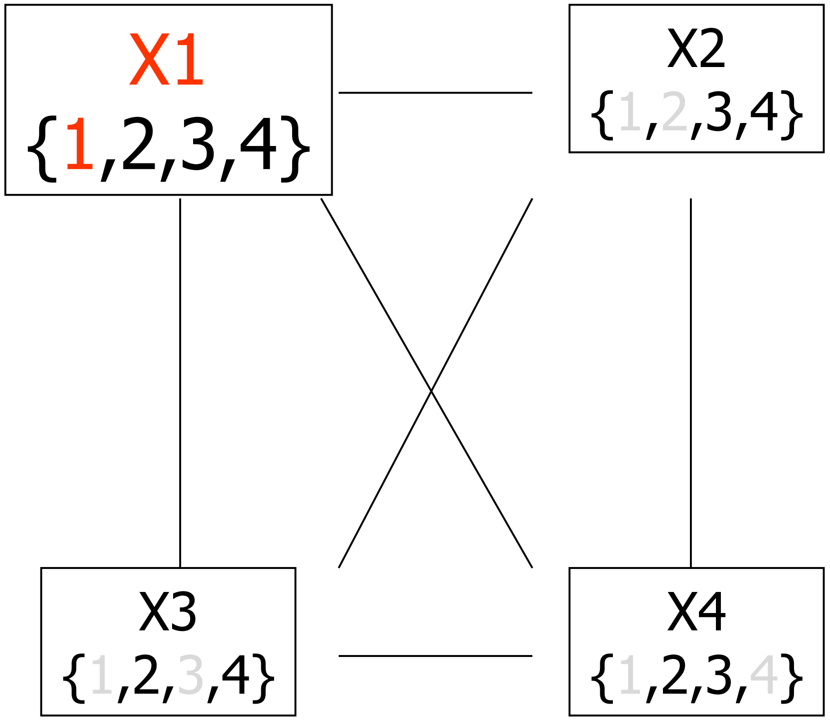
4-Queens Problem



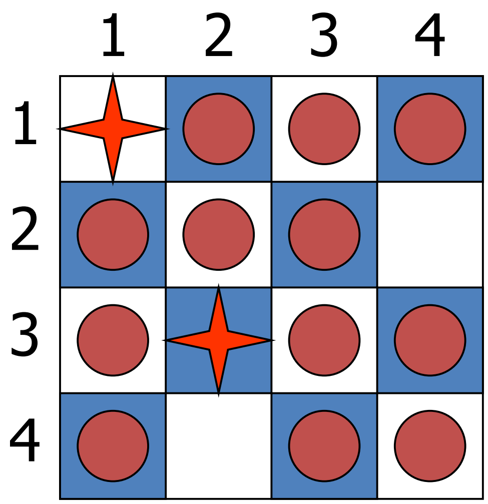


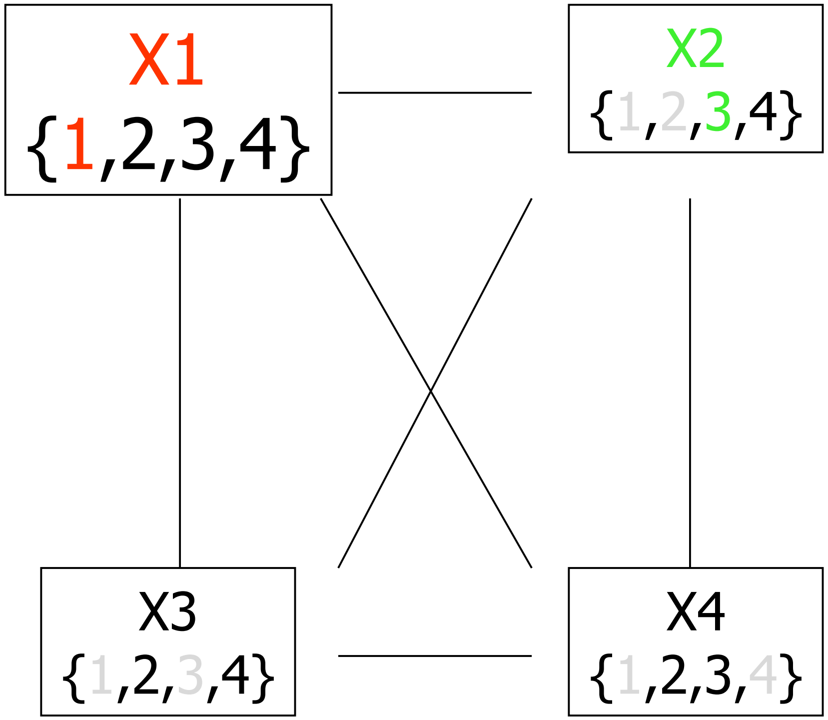
4-Queens Problem



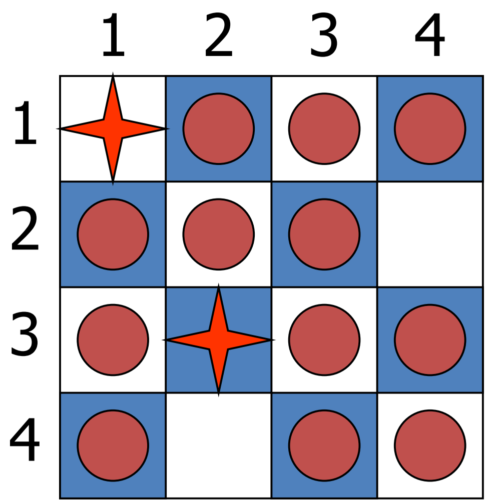


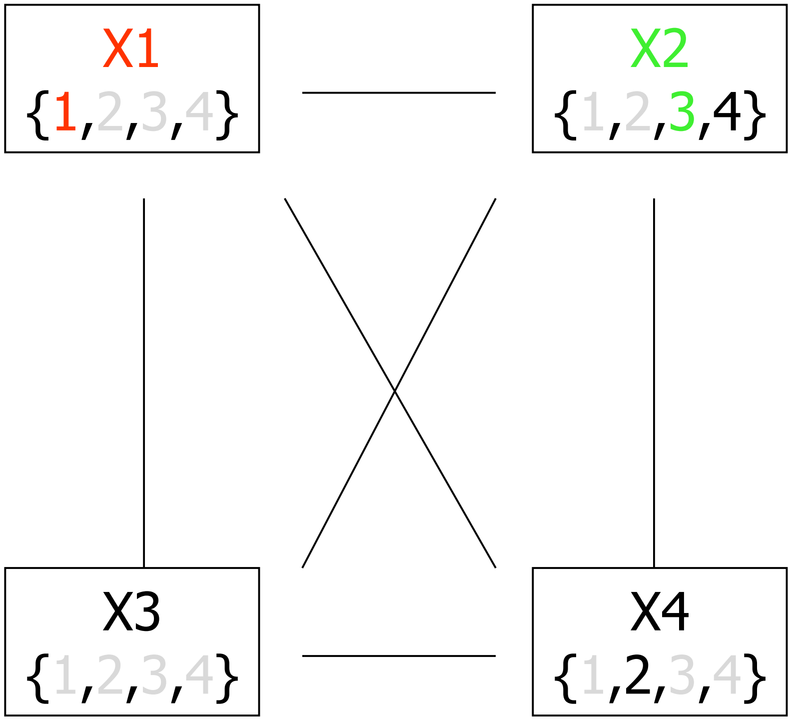
4-Queens Problem



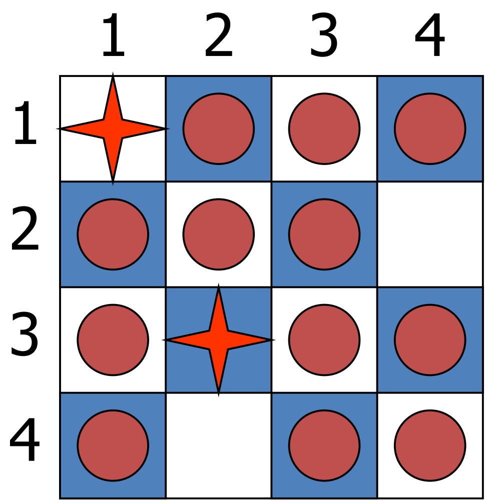


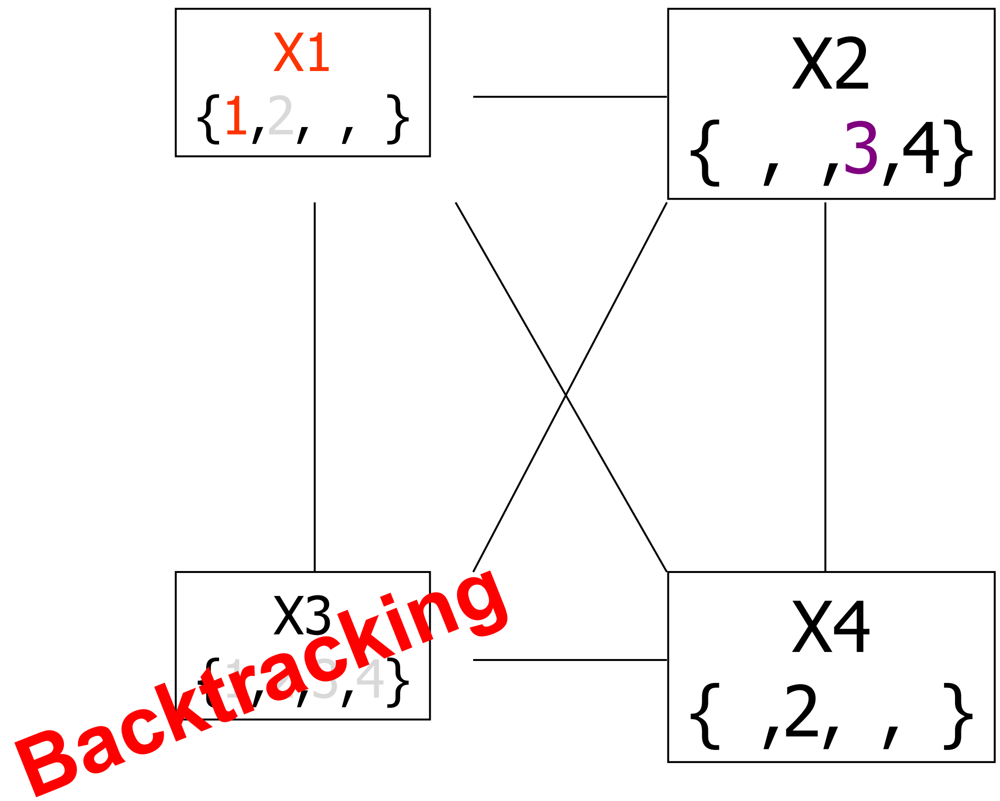
4-Queens Problem



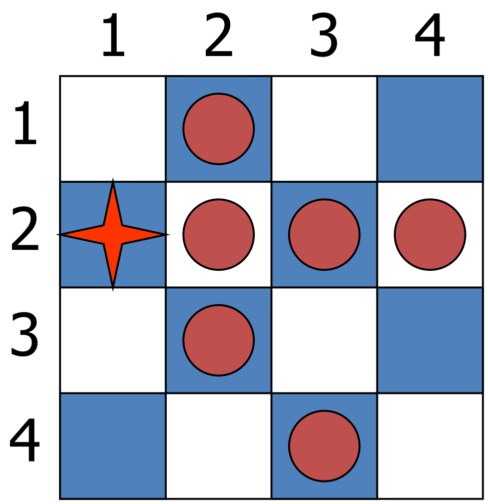


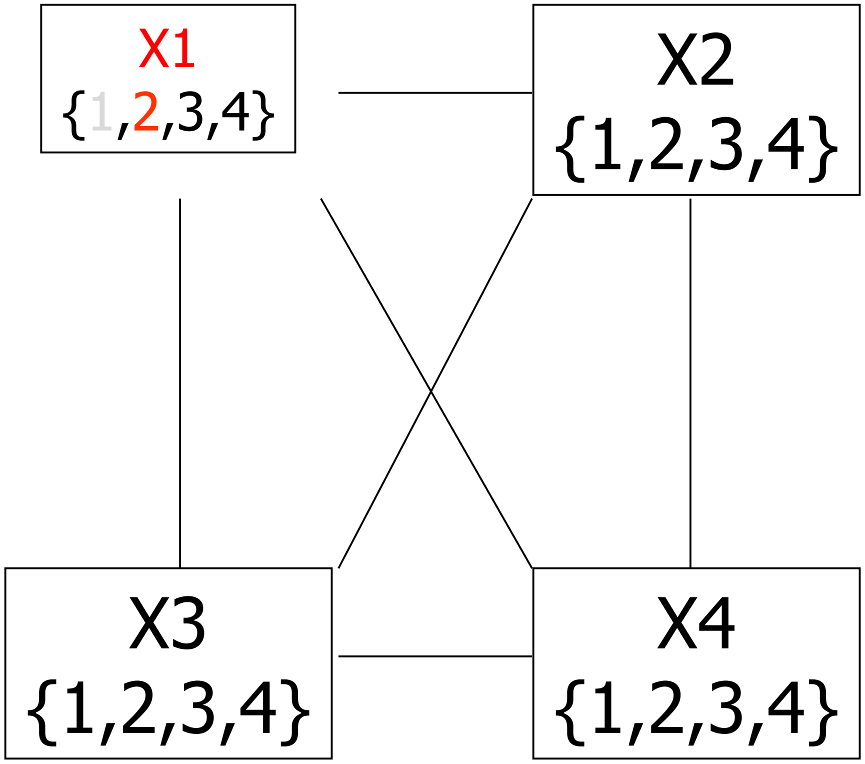
4-Queens Problem



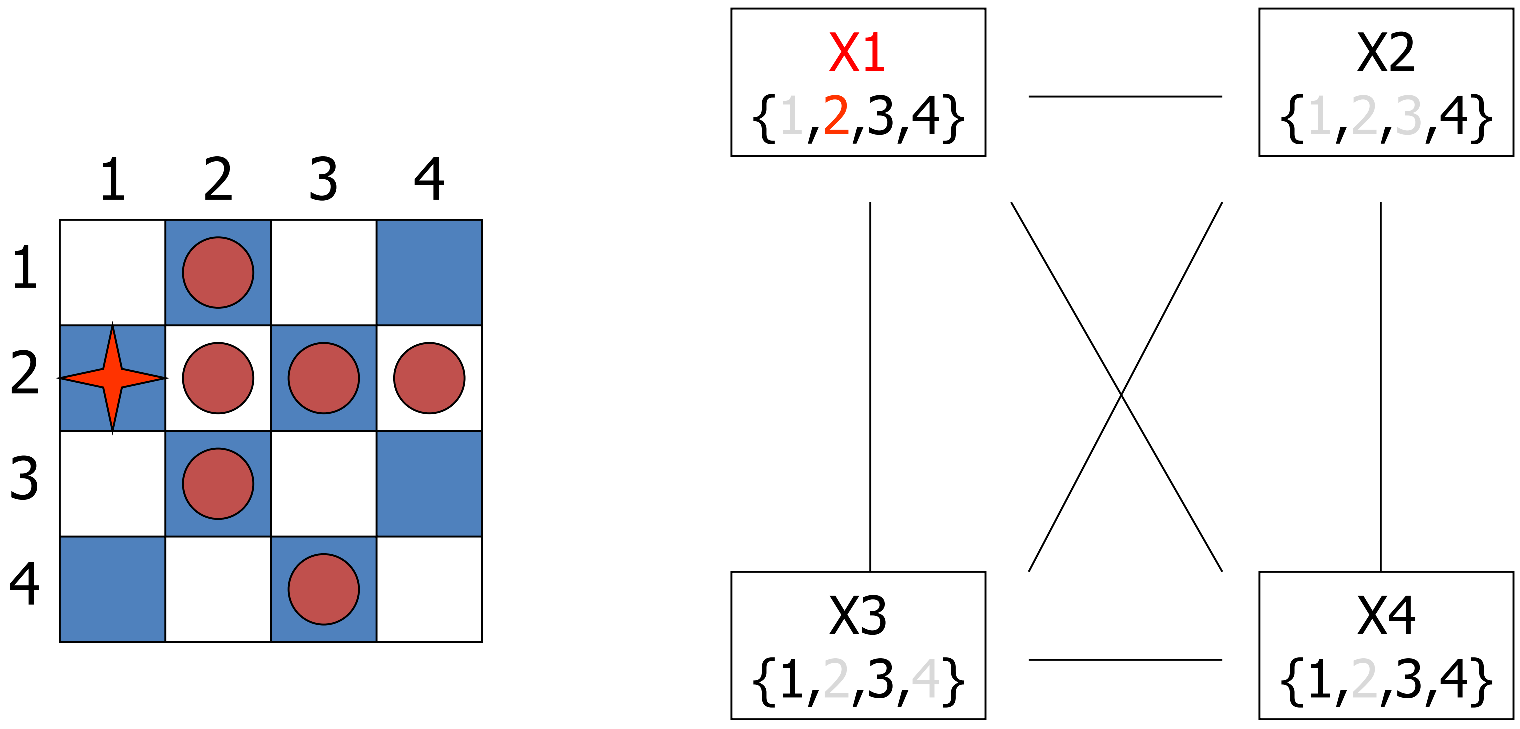


4-Queens Problem

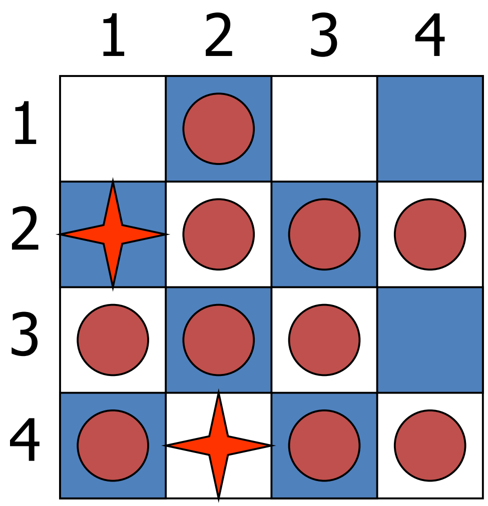


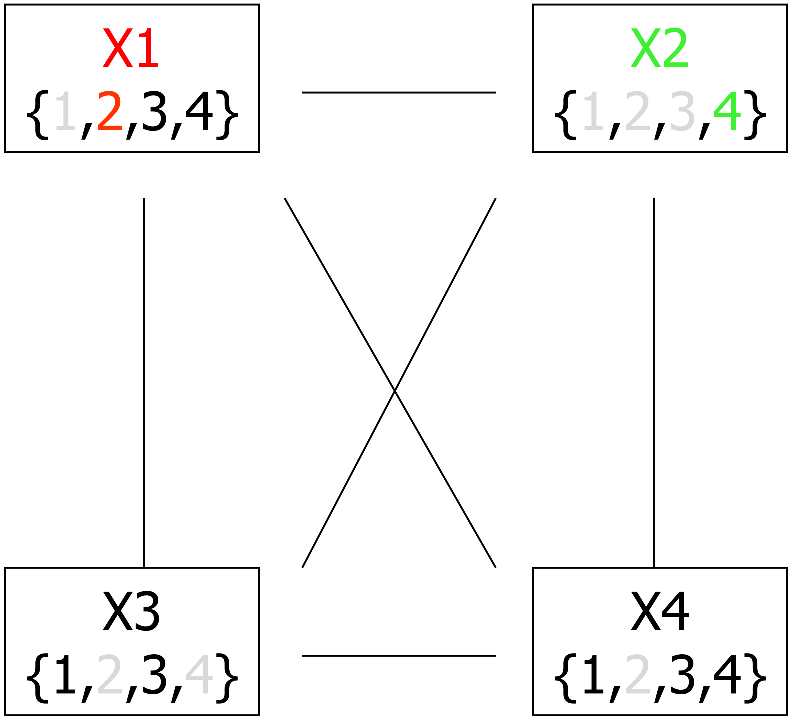


4-Queens Problem

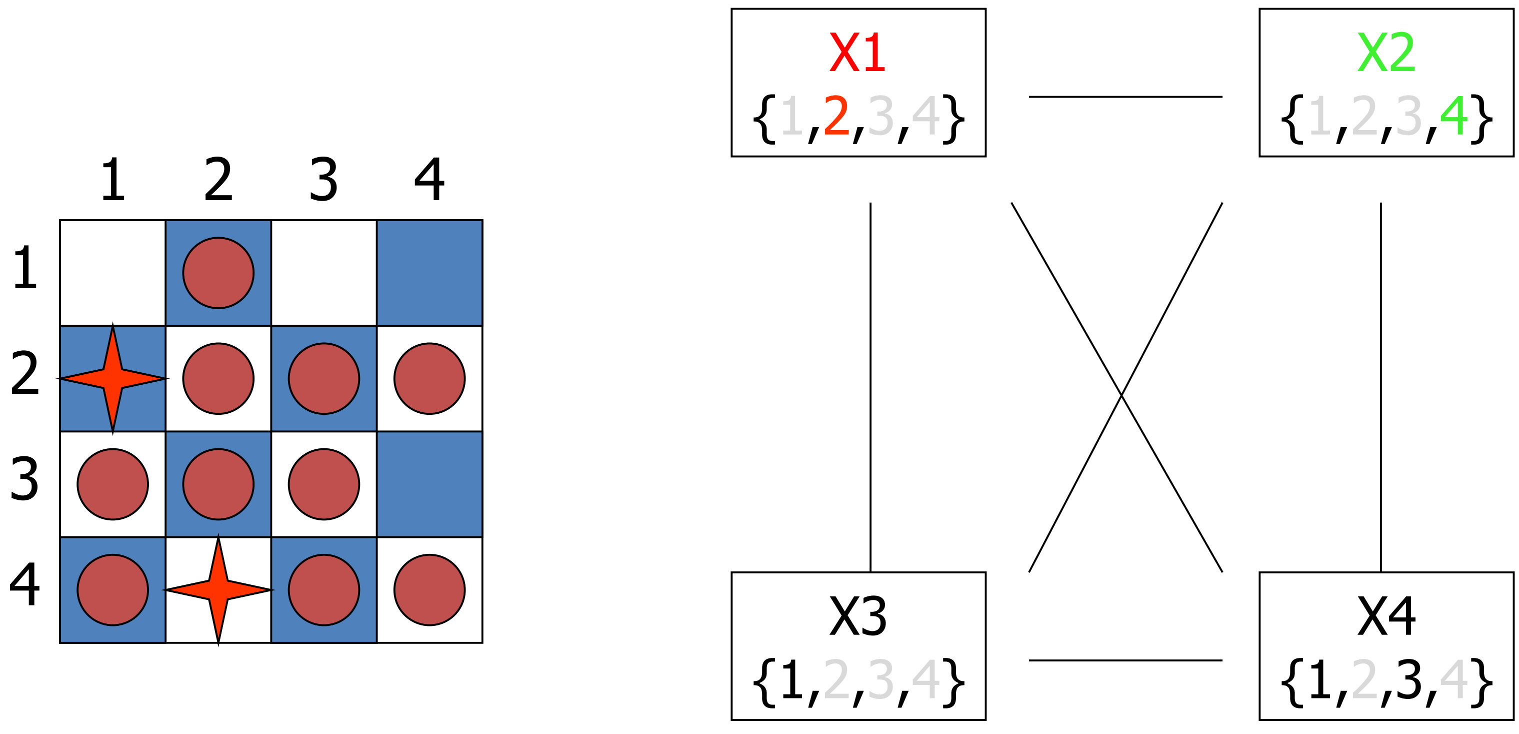


4-Queens Problem

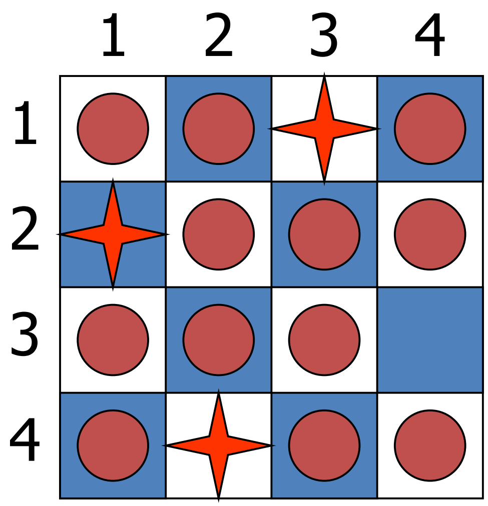


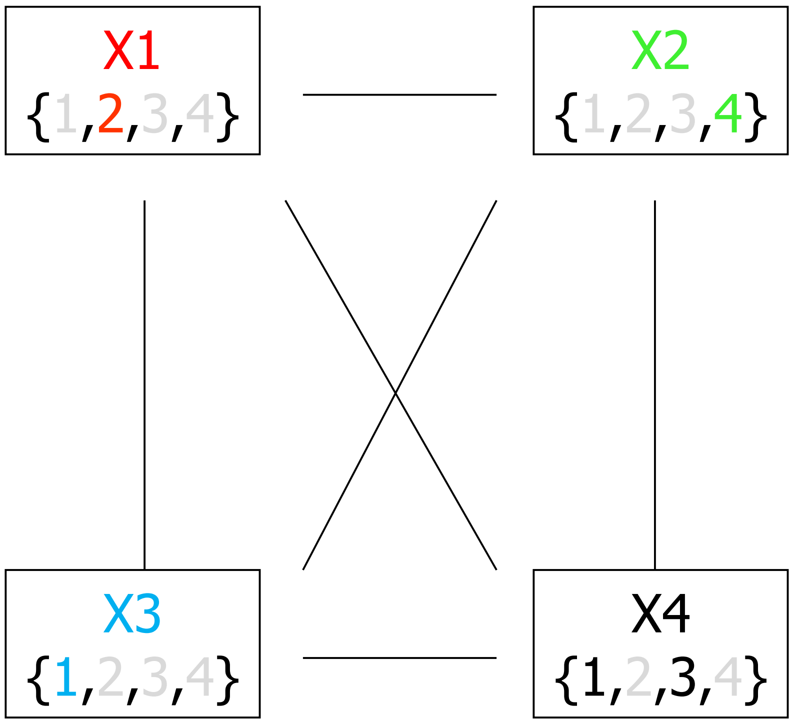


4-Queens Problem

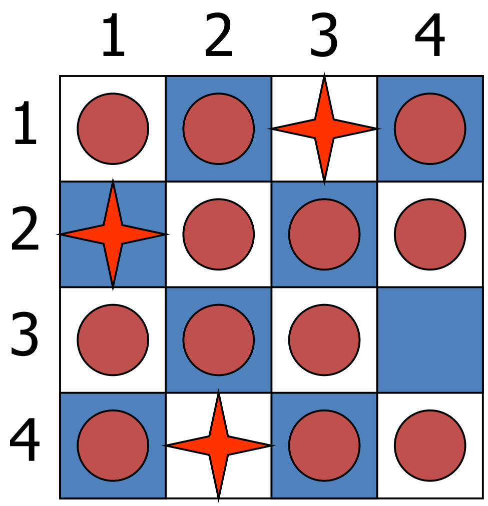


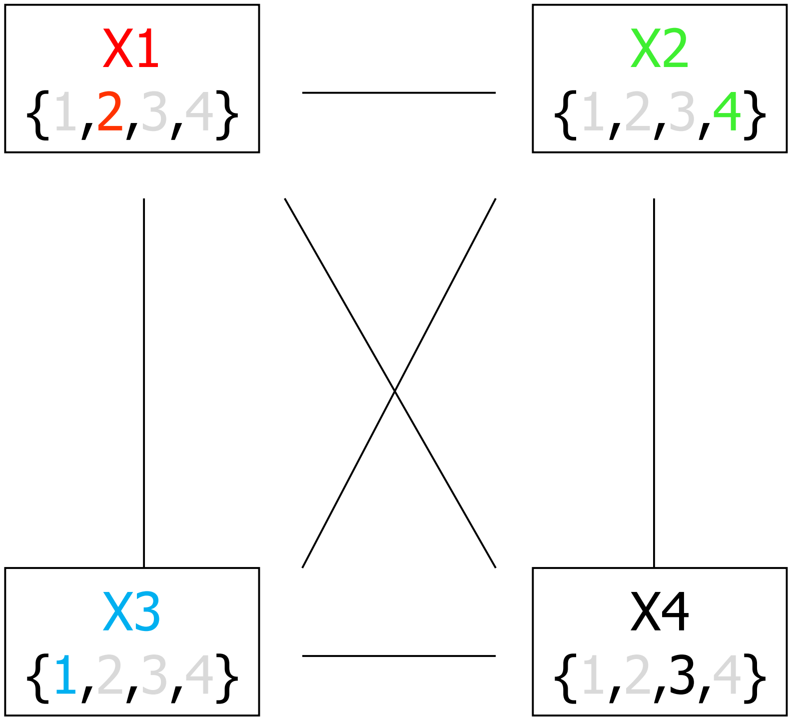
4-Queens Problem





4-Queens Problem





4-Queens Problem