约束满足问题

1 何谓CSP：求解n-qeens、CSP形式化、约束图、约束类型

2 约束传播与局部相容性(结点相容、弧相容、路径相容)

3 CSP形式化为一个搜索问题（回溯法）

4 提高搜索效率（变量顺序、值的顺序、提前检查失败、利用树形结构）

求解n-qeens、CSP形式化、约束图、约束类型

何谓CSP？

Review: local search - hill climbing

- *h* = number of pairs of queens that are attacking each other, either directly or indirectly

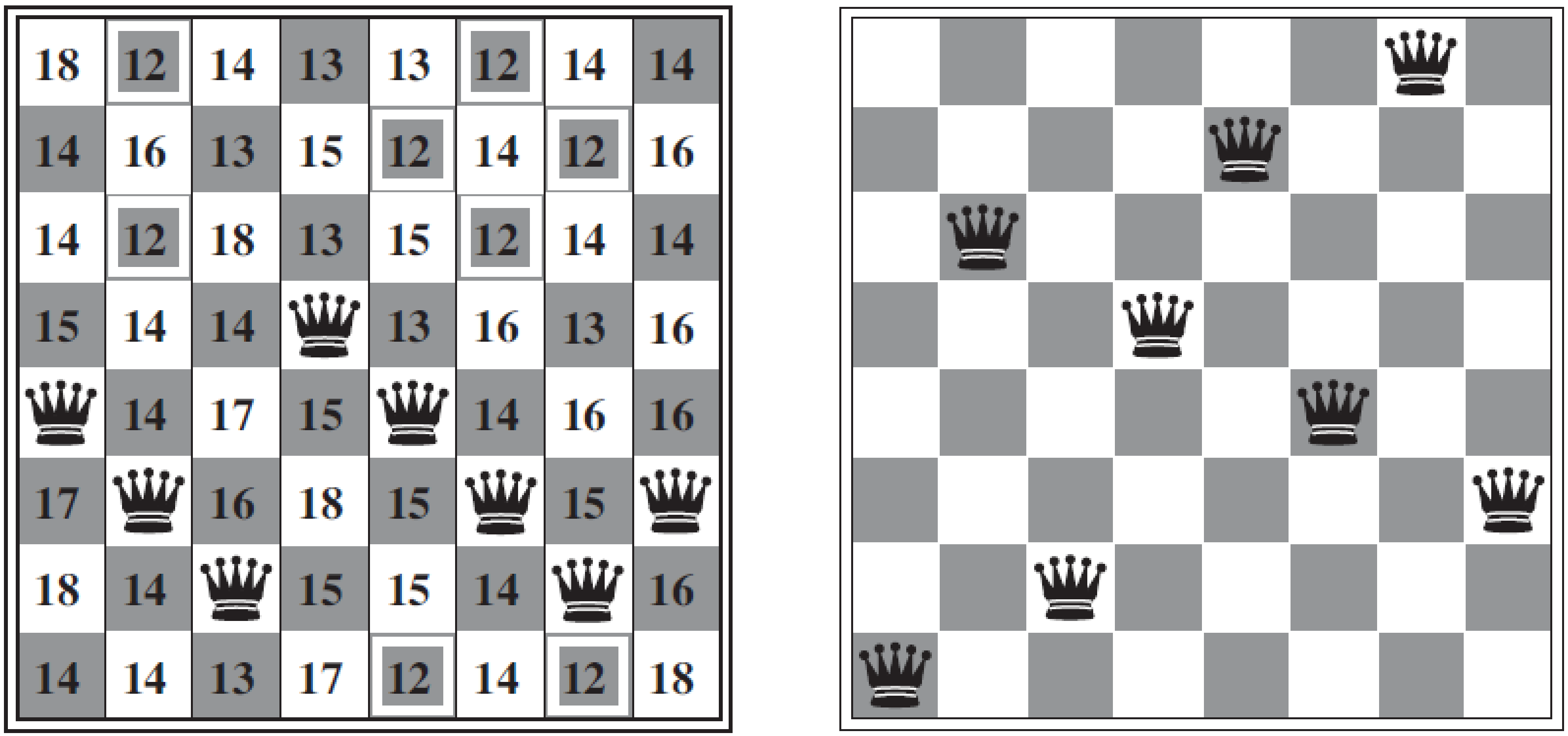


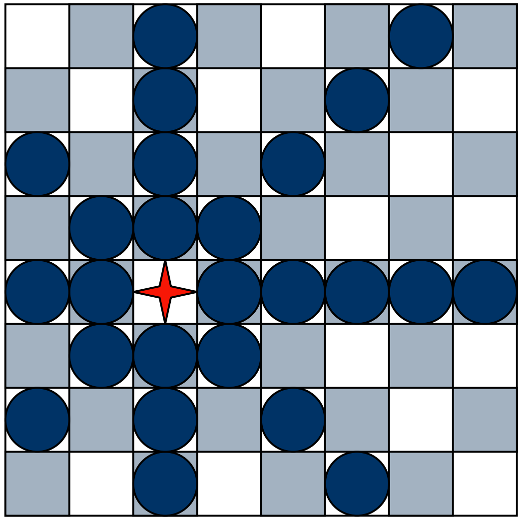
Figure 4.3 (a) An 8-queens state with heuristic cost estimate $h = 17$, showing the value of h for each possible successor obtained by moving a queen within its column. The best moves are marked. (b) A local minimum in the 8-queens state space; the state has $h = 1$ but every successor has a higher cost.

what is CSP?

- 使用要素化表示来描述状态：一组变量，每个变量有自己的值。当每个变量都有自己的赋值同时满足所有关于变量的约束时，问题就得到了解决。这类问题称为**约束满足问题**，简称CSP

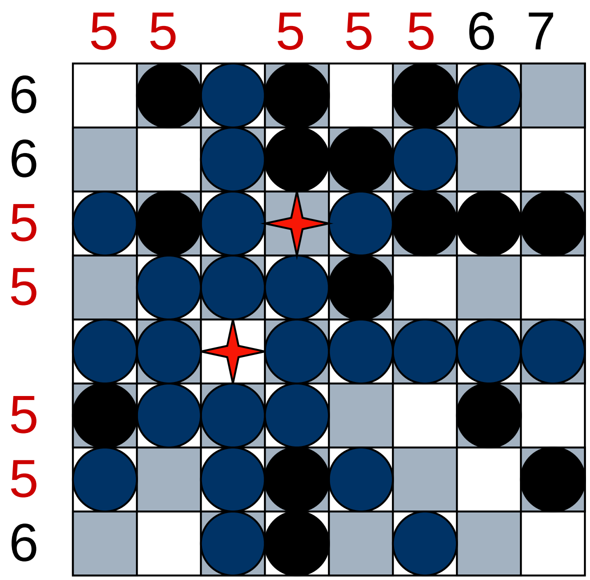
Constraint Propagation

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



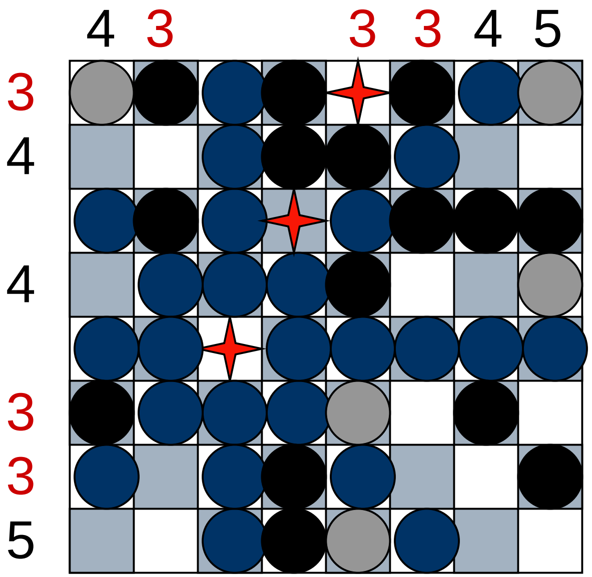
Constraint Propagation

- 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



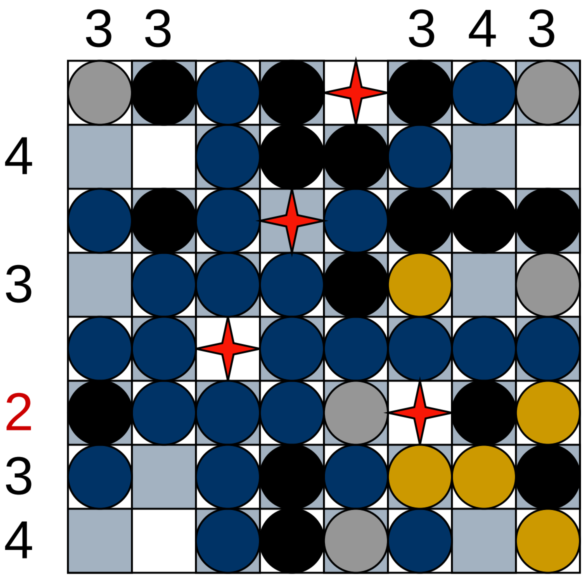
Constraint Propagation

- 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



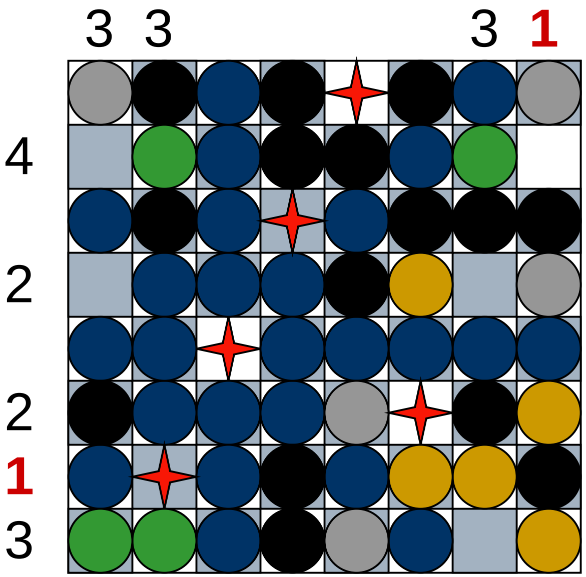
Constraint Propagation

- 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



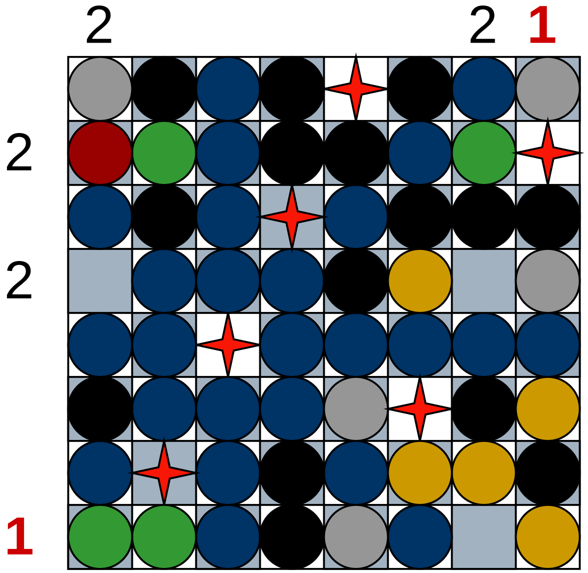
Constraint Propagation

- 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



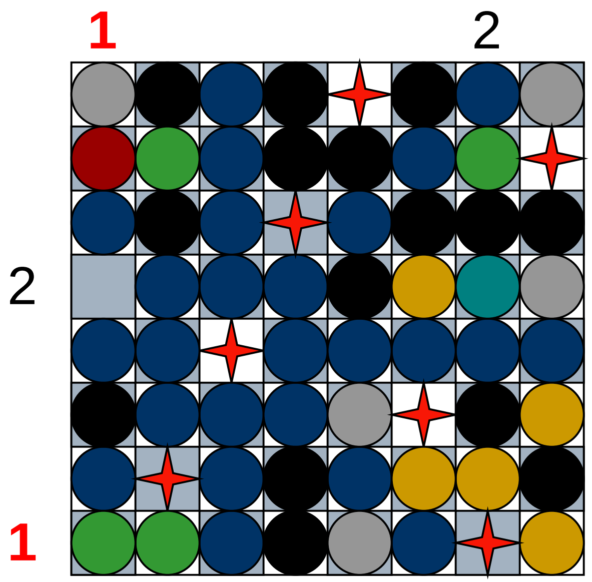
Constraint Propagation

- 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



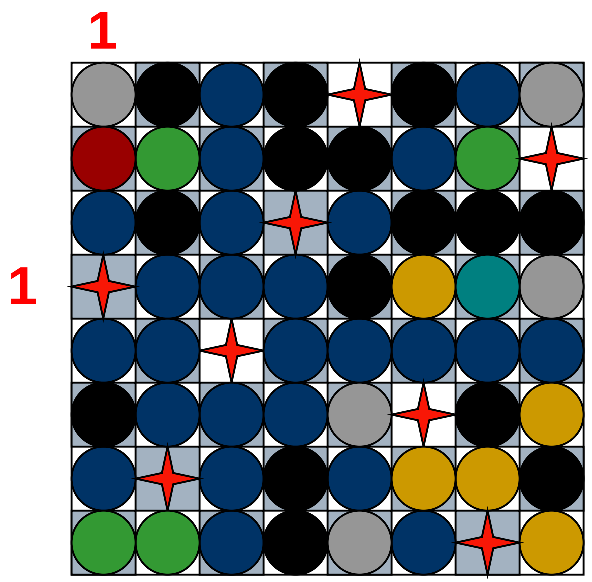
Constraint Propagation

- 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



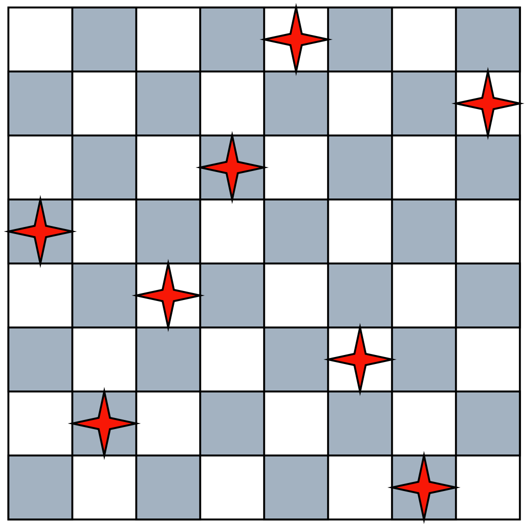
Constraint Propagation

- 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

- 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



What’s the difference with local search ?

Constraint Propagation vs. Local Search

- local search: full assignment
- constraint propagation: partial assignment

how to formalize a CSP ?

What are Constraint Satisfaction Problems ? {6.1}

- a CSP consists of
  - a set of variables, X={X_1, …, X_n }.
  - a set of domains, D={D_1, …, D_n }.
  - a set of constraints, C.
    Each constraint C_i consists of <scope, rel>.
    scope is a tuple of variables, and *rel* defines the relation between those variables.

- Solutions are complete and consistent assignments

example: 8-queens{6.1}

- CSP formulation of 8-queens
- Variables: ______________________
- Domain: ________________________
- Constraints:_____________________
- a solution {5,2,4,6,8,3,1,7}

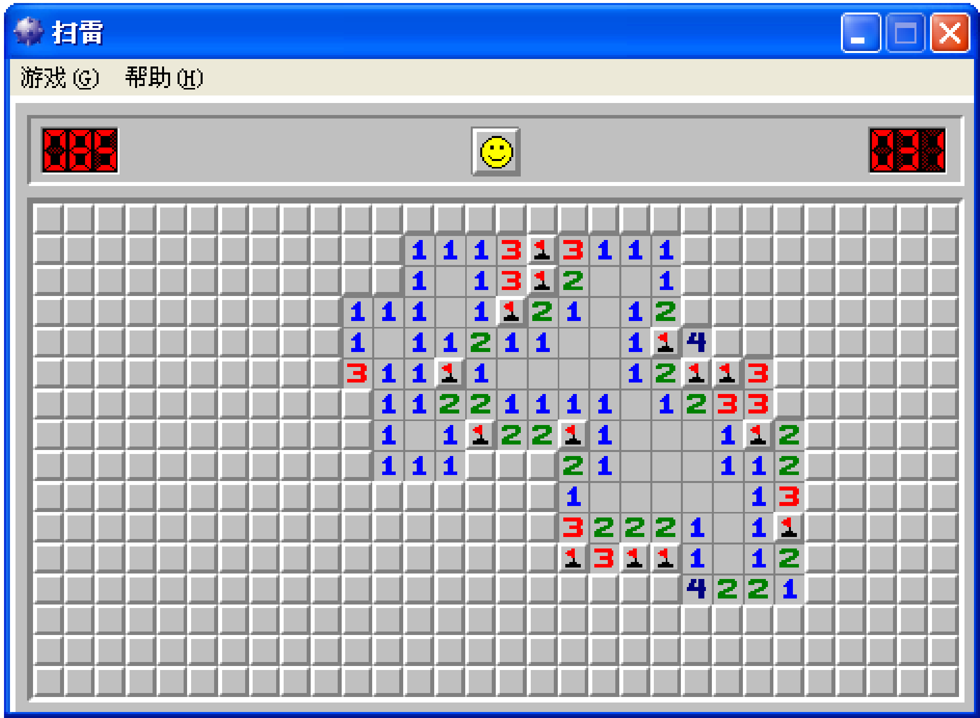
example: 8-queens{6.1}

- CSP formulation of 8-queens
- Variables: $X=\{X1,X2,X3,\ldots,X8\}$
- Domain: $D=\{D1,D2,D3,\ldots,D8\}$
  $Di=\{1,2,3,4,5,6,7,8\}$
- Constraints: $C=\{<\{Xi,Xj\}, Xi \neq Xj \; \& \; |Xi-Xj| \neq |i-j|> \mid i \neq j, 1 \leq i \leq 8, 1 \leq j \leq 8\}$

- a solution {5,2,4,6,8,3,1,7}

Example: Minesweeper Game

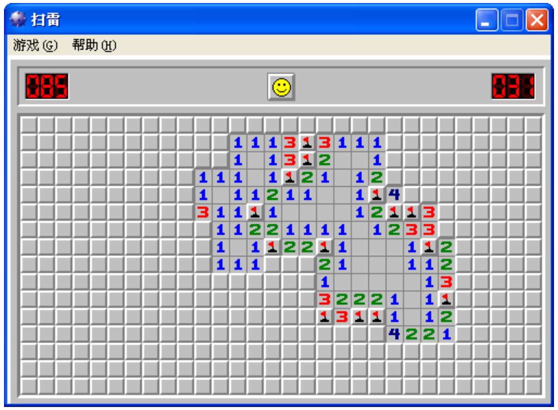
- Each square contains either a zero (touching no bombs in its eight neighboring squares); a number n (touching exactly n bombs); or nothing (unknown).
- variables?
- domain?
- constraints?
- solution?



Example: Minesweeper Game

- Variables: $X=\{X_{ij} \mid i=1,2,\ldots,\ j=1,2,\ldots\}$
- Domain:
  - $D=\{D_{ij} \mid i=1,2,\ldots,\ j=1,2,\ldots\}$
  - $D_{ij}=\{0,1,2,\ldots,8,m\}$
- Constraints:

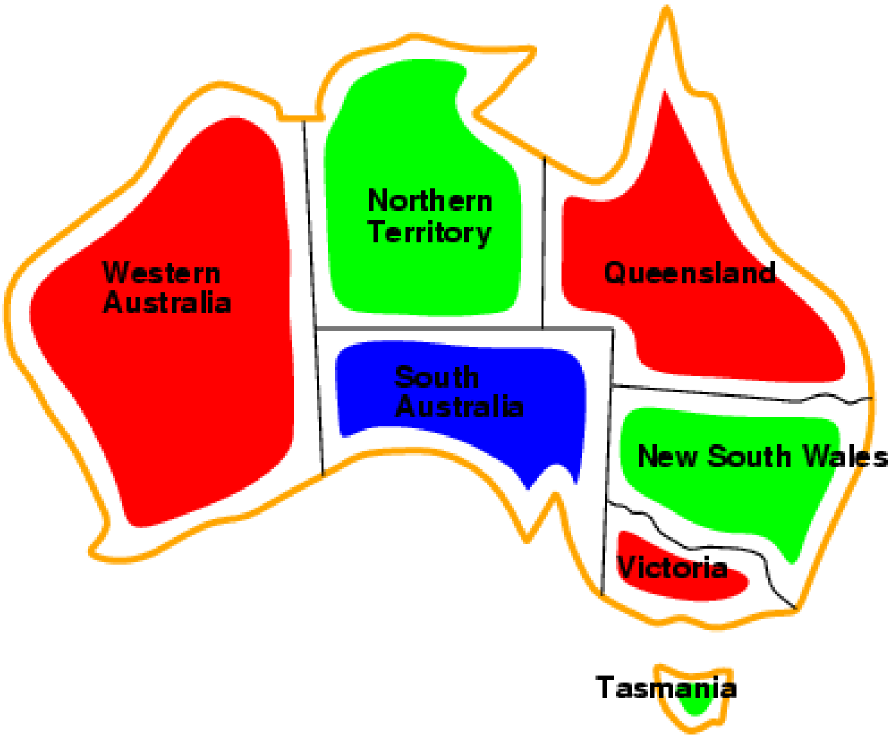
  if Dij=0,1,…,8, then there are Dij mines in the neighboring blocks of block_{ij}.



- Solutions are assignments satisfying all constraints

Example: Map-Coloring {6.1.1}

- color Australia regions using red, green and blue to make adjacent regions have different colors.
- variables?
- domain?
- constraints?
- solution?



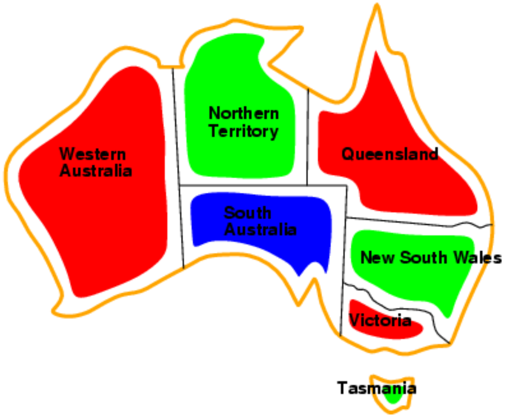
Example: Map-Coloring {6.1.1}

- Variables: $X=\{WA, NT, Q, NSW, V, SA, T\}$
- Domain: $D=\{D_1,\ldots,D_7\}, D_i=\{red, green, blue\}$
- Constraints: adjacent regions must have different colors $C=\{WA\neq NT, WA\neq SA, NT\neq SA, NT\neq Q, SA\neq Q, SA\neq NSW, SA\neq V, Q\neq NSW, NSW\neq V\}$

$SA\neq WA$是< (SA, WA)，$SA\neq WA$>的简洁表示

- Solutions are assignments satisfying all constraints

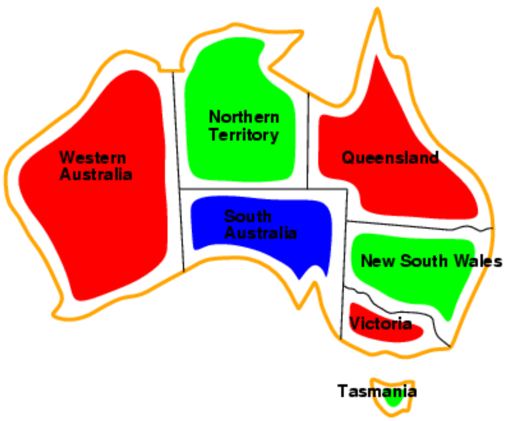
$\{WA=red, NT=green, Q=red, NSW=green, V= red, SA=blue, T= green\}$

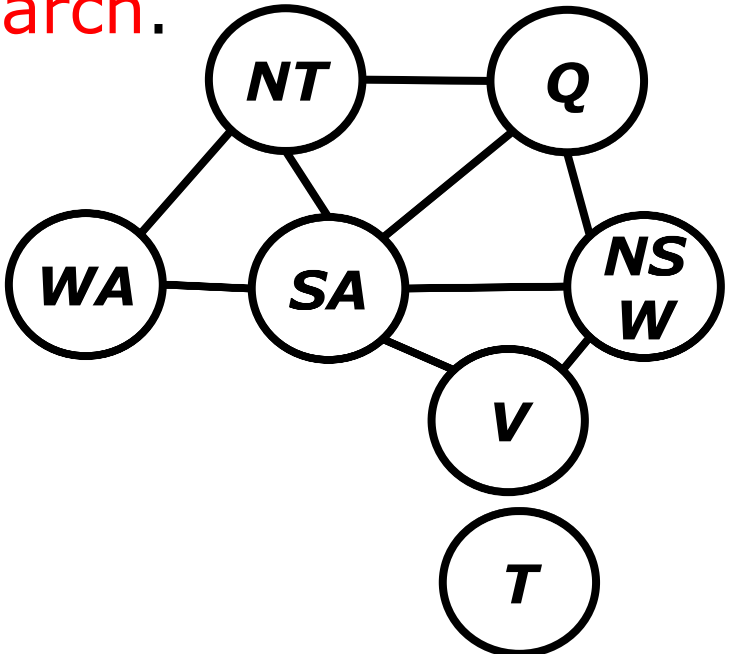


represent constraints as a graph!

Constraint Graph {6.1.1}

- 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.





Varieties of Constraints {6.1.3}

- Unary constraints involve a single variable:

  $$SA \neq green$$

- Binary constraints involve pairs of variables:

  $$SA \neq WA$$

- Higher-order constraints involve 3 or more variables (How to represent them using graph?)
- Preferences (soft constraints)
  - red is better than green
  - Gives constrained optimization problems
  - Representable by a cost for each variable assignment

What is the reasoning in CSP?

结点相容(1-相容)、弧相容(2-相容)、路径相容(3-相容)、K-相容

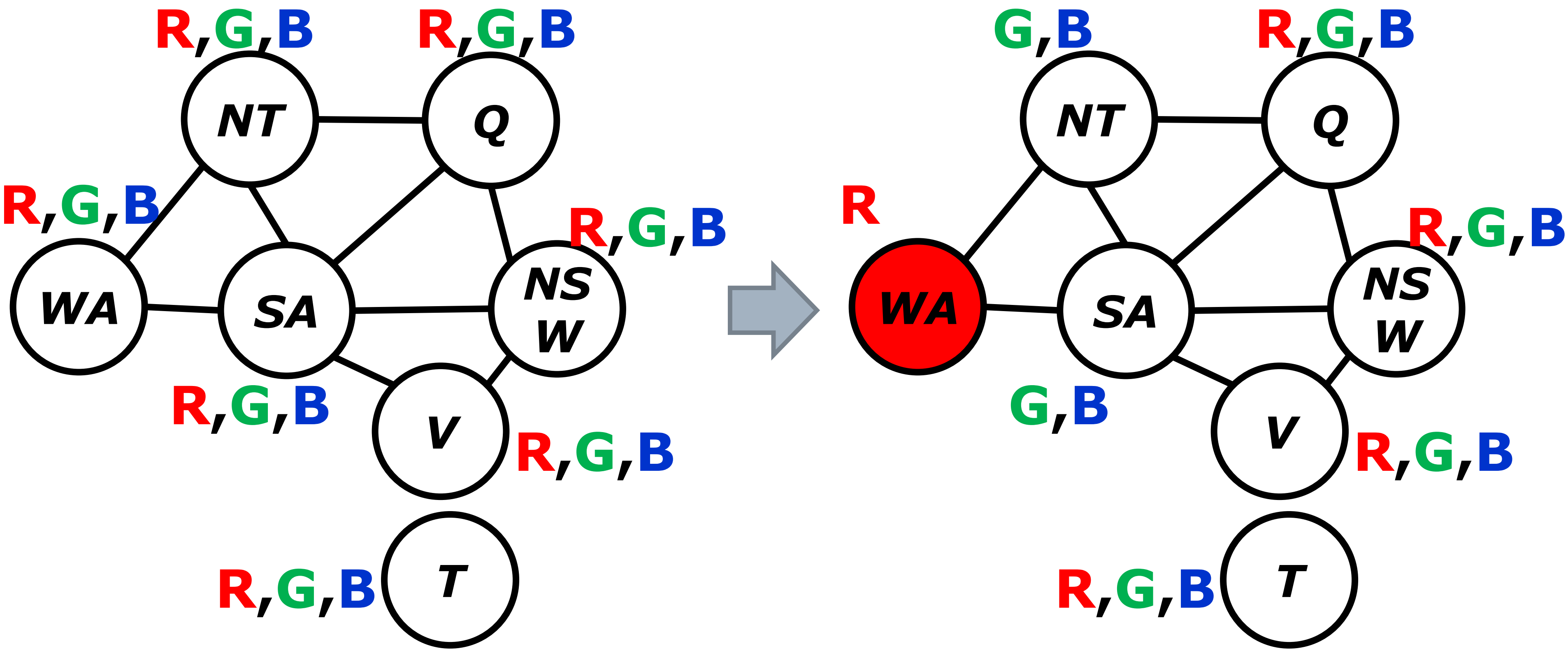
约束传播与局部相容性

constraint propagation {6.2}

- CSP中的推理：约束传播。约束传播使用约束减小一个变量的合法取值范围，从而影响到跟此变量有约束关系的另一变量的取值。
  - 给一个变量赋值后，则通过约束传播可缩小与其有约束关系的其他变量的值域。

constraint propagation {6.2}

- 若WA赋值R，则通过约束传播，NT和SA的值域中删除R

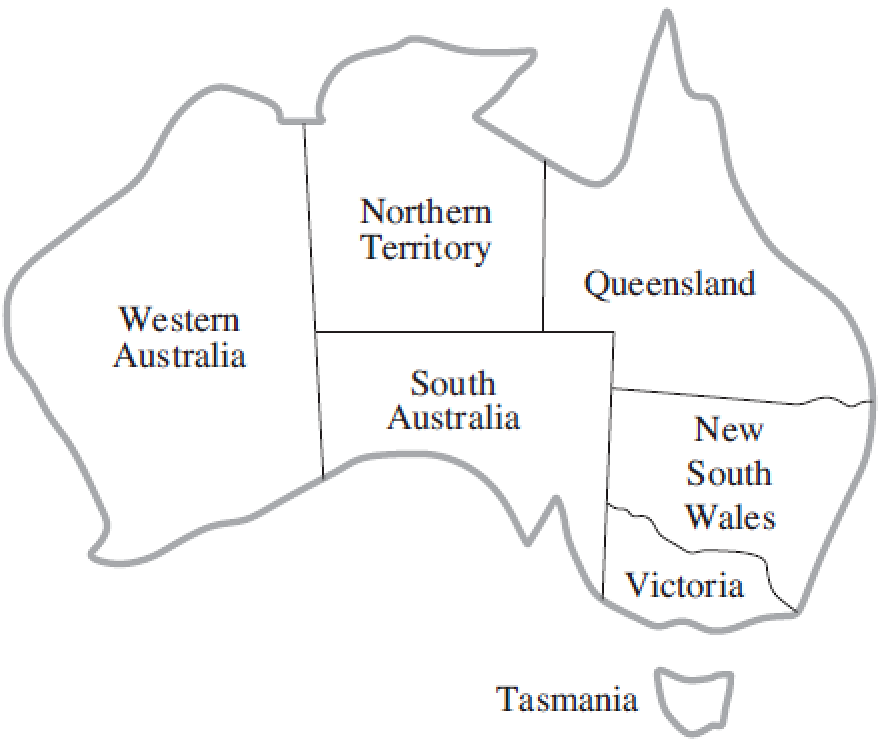


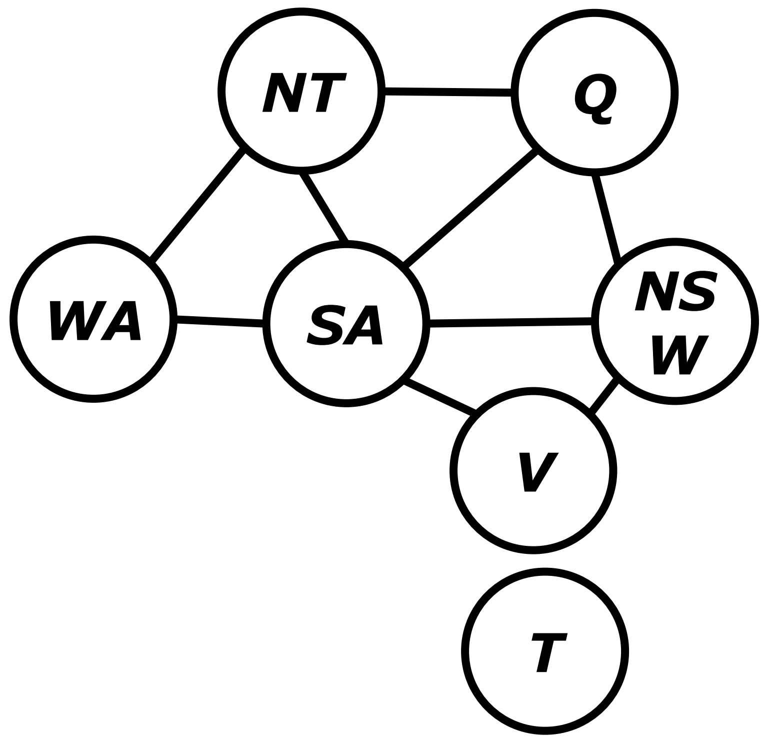
constraint propagation {6.2}

- 在CSP中,算法可以进行约束传播，也可以搜索（从几种可能性中选择新的变量赋值）。约束传播与搜索可以交替进行，或者也可以将约束传播作为搜索前的预处理步骤。

constraint propagation {6.2}

- 约束传播的核心思想是**局部相容性**。
- 增强约束图中各部分局部相容性会导致不相容的结点取值被删除。



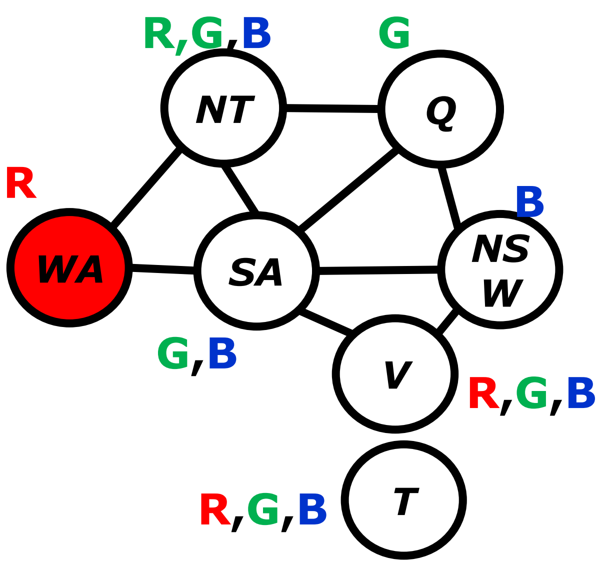


Node consistency {6.2.1}

- 如果单个变量（对应于CSP网络中的结点）值域中的所有取值满足它的一元约束，就称此变量是**结点相容**的。例如，如果地图着色问题中南澳洲人不喜欢绿色，变量*SA*原来值域为{*red*, *green*, *blue*}，删除*green*此结点即为结点相容的，此时*SA*的值域为{*red*, *blue*}。如果网络中每个变量都是结点相容的，则此网络是结点相容的。

arc consistency {6.2.2}

- 对于变量X_i、X_j，若对X_i的每个赋值在X_j都存在某个取值满足弧（X_i, X_j）的二元约束，则称X_i对于X_j是弧相容的。



哪些是弧相容的?

(1) WA对于SA

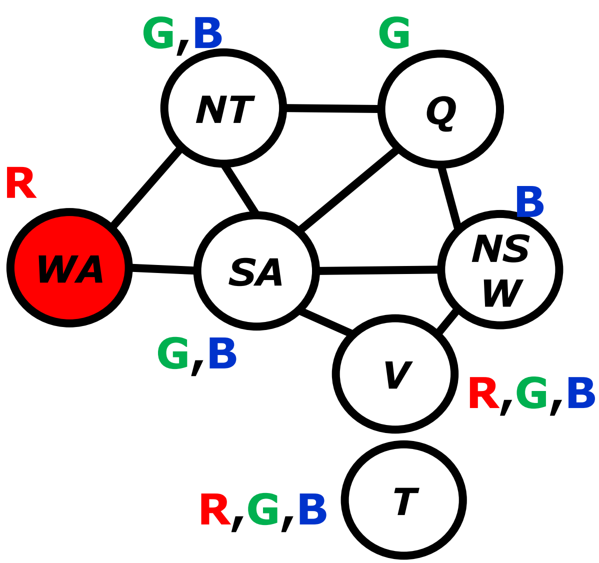
(2) WA对于NT

(3) SA对于Q

(4) SA对于NSW

arc consistency {6.2.2}

- 对于变量X_i、X_j，若对X_i的每个赋值在X_j都存在某个取值满足弧（X_i, X_j）的二元约束，则称X_i对于X_j是弧相容的。



哪些是弧相容的？

(1) WA对于SA yes

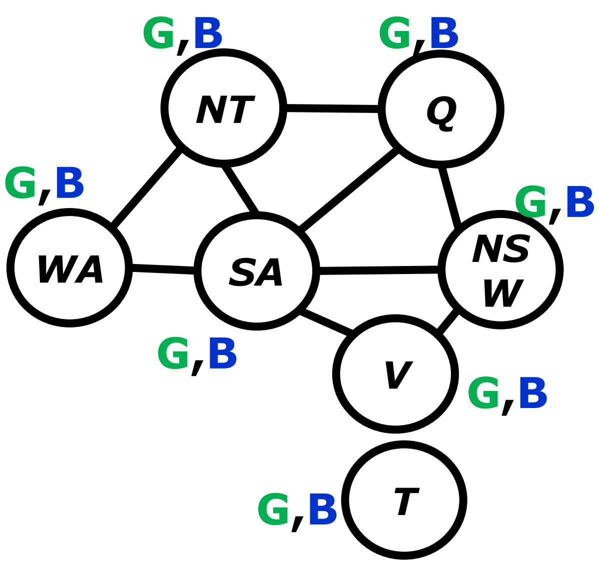
(2) WA对于NT yes

(3) SA对于Q no

(4) SA对于NSW no

arc consistency {6.2.2}

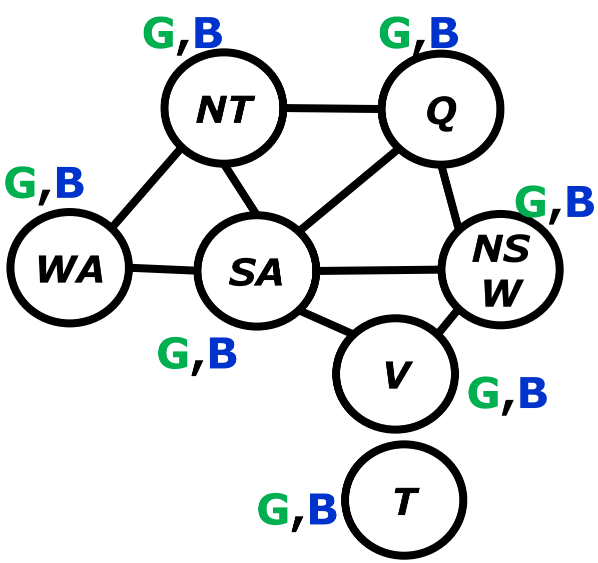
- 对于变量X_i、X_j，若对X_i的每个赋值在X_j都存在某个取值满足弧（X_i, X_j）的二元约束，则称X_i对于X_j是弧相容的。如果每个变量对于其它变量都是弧相容的，则称该网络是弧相容的



是否是网络相容的？是否有解？

arc consistency {6.2.2}

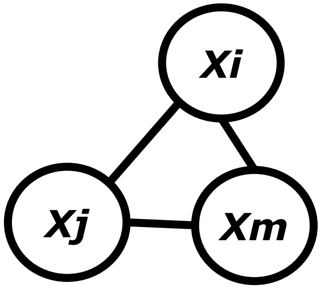
- 对于变量X_i、X_j，若对X_i的每个赋值在X_j都存在某个取值满足弧（X_i, X_j）的二元约束，则称X_i相对X_j是弧相容的。如果每个变量相对其它变量都是弧相容的，则称该网络是弧相容的



是网络相容的，但显然此题无解。路径相容可以检测出无解。

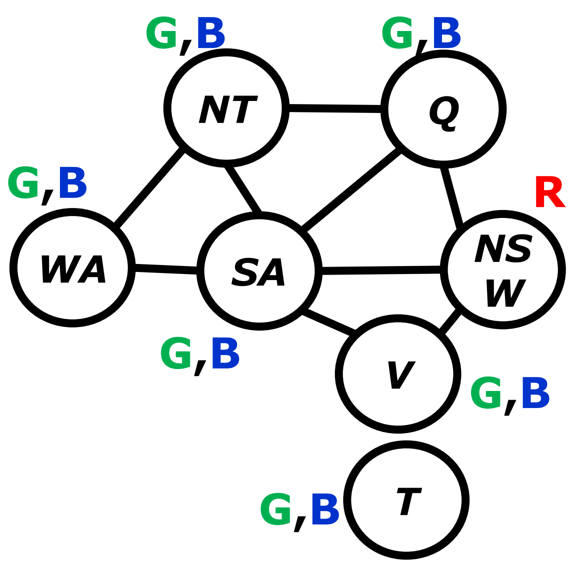
path consistency {6.2.3}*

- 对$\{X_i, X_j\}$的每一个相容赋值$\{X_i=a, X_j=b\}$，X_m都有合适的取值同时使得$\{X_i, X_m\}$和$\{X_m, X_j\}$是相容的，则称集合$\{X_i, X_j\}$对于X_m是（路径）相容的。被称为**路径相容**，是因为这很像是一条从X_i途经X_m到X_j的路径。



path consistency {6.2.3}*

- 对$\{X_i, X_j\}$的每一个相容赋值$\{X_i=a, X_j=b\}$，X_m都有合适的取值同时使得$\{X_i, X_m\}$和$\{X_m, X_j\}$是相容的，则称集合$\{X_i, X_j\}$对于X_m是（路径）相容的。被称为**路径相容**，是因为这很像是一条从X_i途经X_m到X_j的路径。



哪些是路径相容的？

(1) {WA, NT} 对于SA

(2) {SA, Q} 对于NSW

(3) {SA, NSW} 对于V

k-consistency {6.2.4}*

- 如果对于任何$k-1$个变量的任何相容赋值，任何第k个变量总能被赋予一个和前$k-1$个变量相容的值，那么这个CSP是k相容的。

- 1-相容是节点相容。2-相容即为弧相容。对二元约束网络，3-相容是路径相容。

Global Constraints {6.2.5}

- 全局约束可涉及任意个约束变量（不一定是所有变量）。例如，*Alldiff*约束表示所有相关变量必须取不同的值。*Alldiff*约束的不相容检测的一种简单形式：如果约束涉及m个变量，可能的不同取值有n个，且$m>n$，那么约束不可能满足。
- 另一个重要的高阶约束是**资源约束**，有时称为*atmost*约束。例如在调度问题中，用P_1，…，P_4表示执行四项任务的人数。总人数不超过10人的约束记为$atmost(10, P_1, P_2, P_3, P_4)$。

- 在CSP中,算法可以进行约束传播，也可以搜索（从几种可能性中选择新的变量赋值）。

CSP形式化为搜索问题:回溯搜索

Standard Search Formulation {6.3}

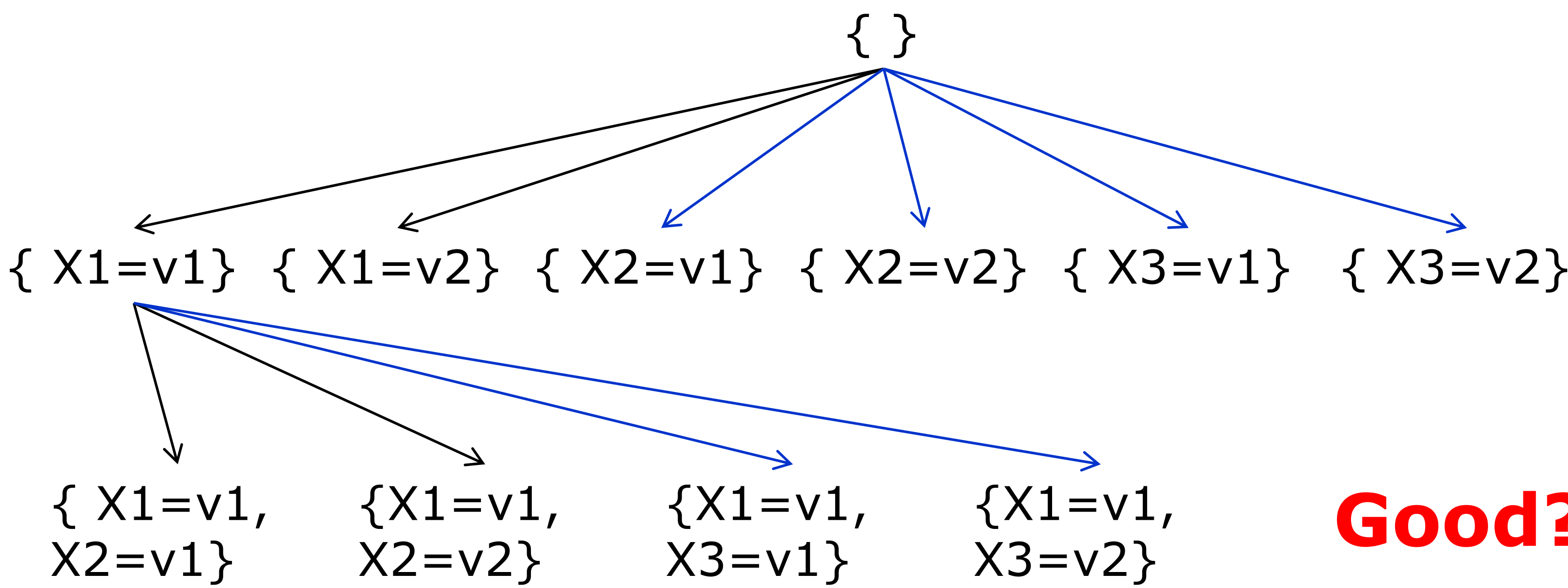
- States are ______________________________
- Initial state: ________________________________
- Transition model
  - ____
- Goal test: the current assignment is complete and satisfies all constraints

Standard Search Formulation {6.3}

- States are defined by the values assigned so far
- Initial state: the empty assignment, {}
- Transition model
  - assign a value to an unassigned variable
  - fail if no legal assignment
- Goal test: the current assignment is complete and satisfies all constraints

Backtracking Search {6.3}

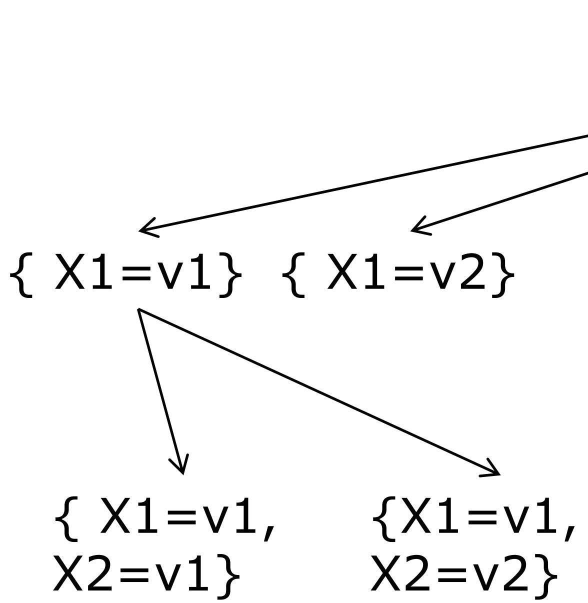
- suppose there are 3 variables, and each has 2 values.



Good??

Backtracking Search {6.3}

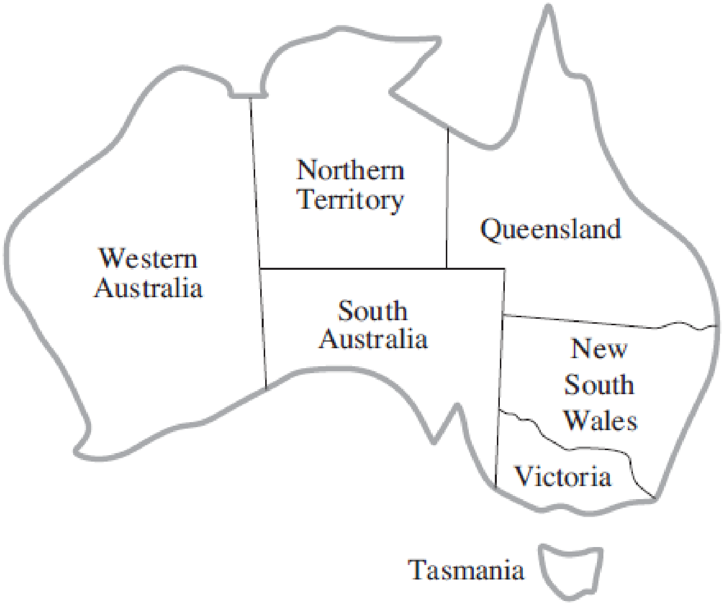
- suppose there are 3 variables, and each has 2 values.

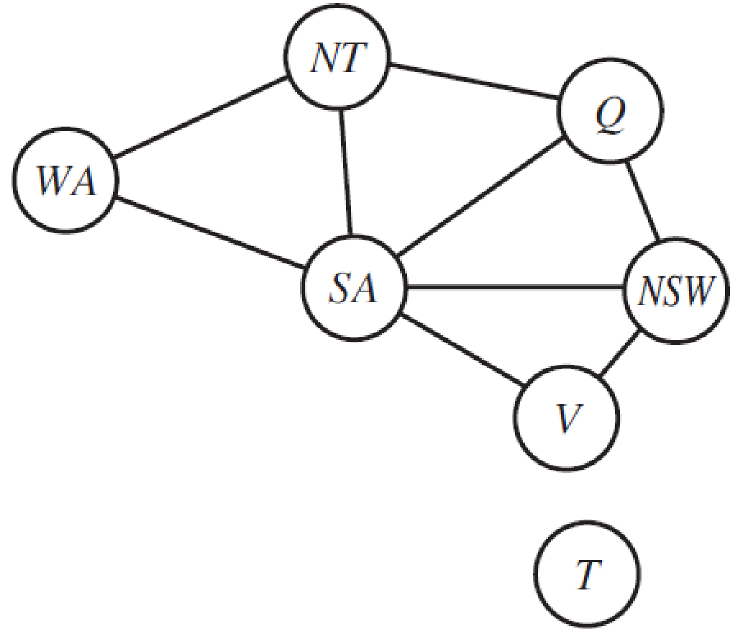


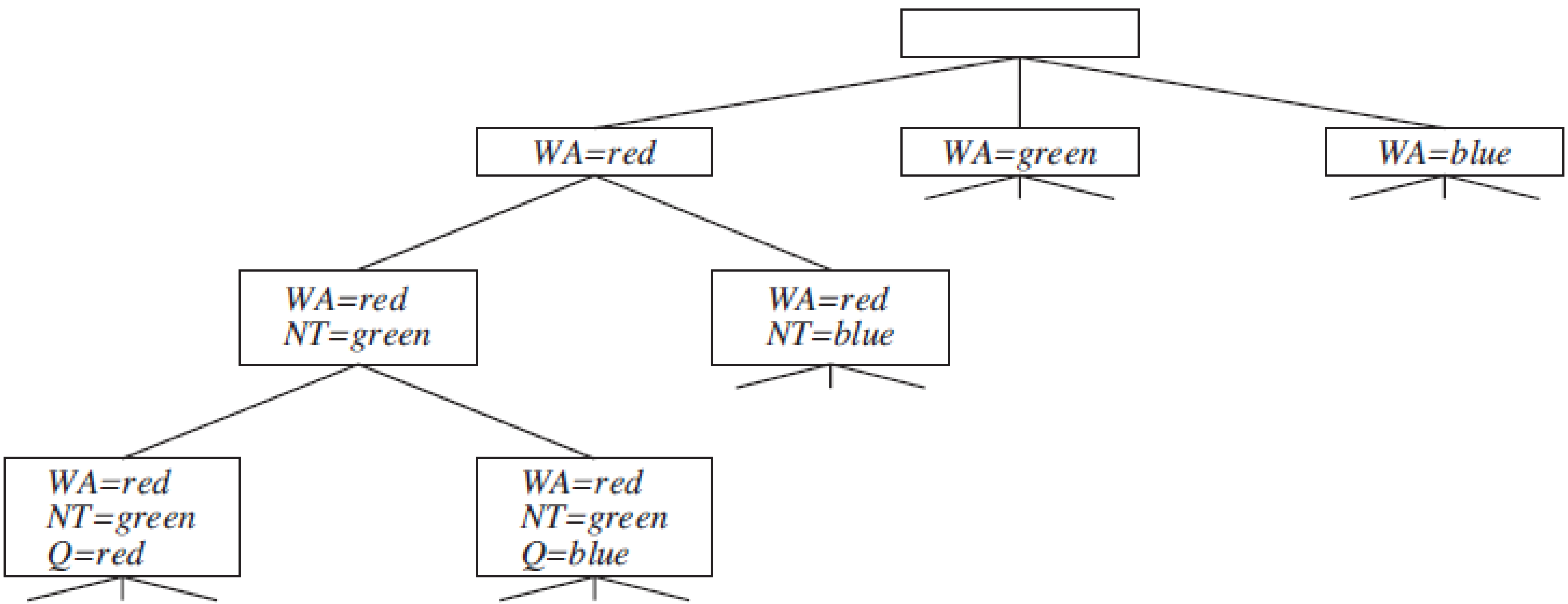
Variable assignments are **commutative.** Only need to consider assignments to a single variable at each step

Backtracking Search {6.3}

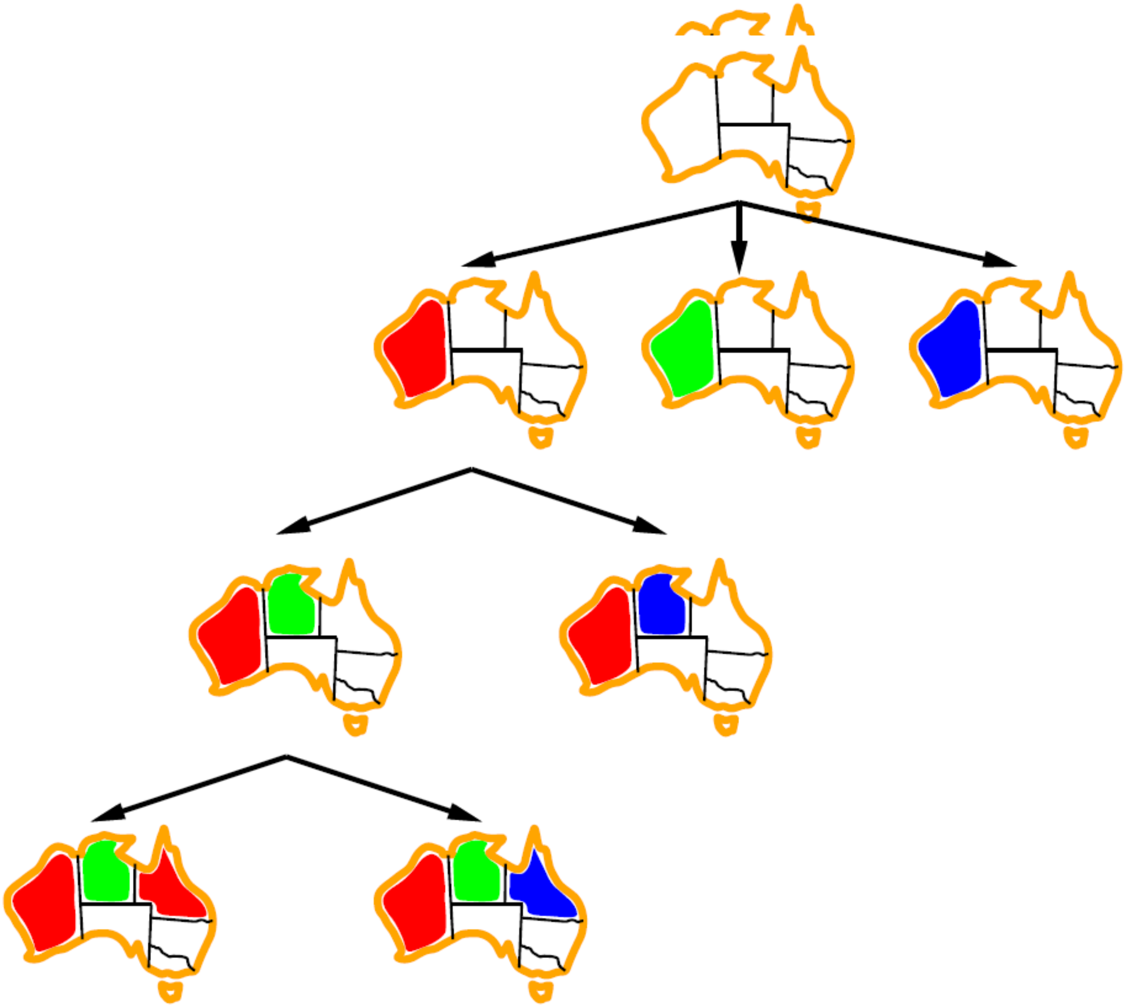
- Idea 1: Only consider a single variable at each point:
  - Variable assignments are **commutative**
    [WA = red then NT = green] same as
    [NT = green then WA = red]
  - Only need to consider assignments to a single variable at each step
- Idea 2: Only allow legal assignments at each point
  - consider only values which do not conflict previous assignments, might have to do some computation
- Depth-first search for CSPs with these two improvements is called backtracking search


Northern Territory
Western Australia
Queensland
South Australia
New South Wales
Victoria
Tasmania


NT
Q
WA
SA
NSW
V
T


WA=red
WA=green
WA=blue
WA=red
NT=green
WA=red
NT=blue
WA=red
NT=green
Q=red
WA=red
NT=green
Q=blue

Backtracking Example {+}



从哪些方面入手
可提高搜索效率?

变量顺序（最少剩余值、最大度）、值的顺序（最少约束值）、提前检查失败（前向检验、维护弧相容）、利用树形结构

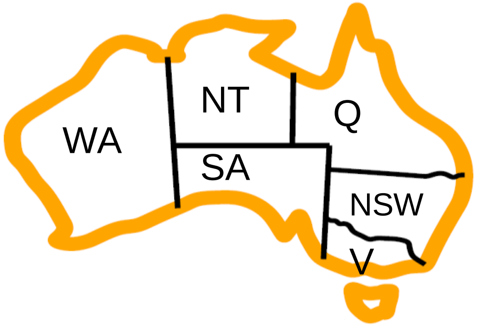
提高搜索效率

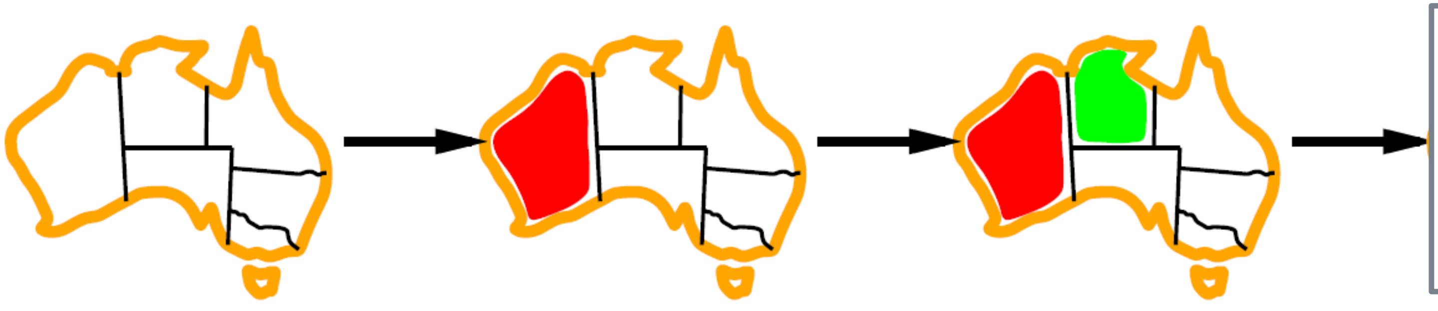
Improving Backtracking {6.3}

- General-purpose ideas can give huge gains in speed:
  - Which variable should be assigned next?
  - In what order should its values be tried?
  - Can we detect inevitable failure early?
  - Can we take advantage of problem structure?

which variable first?

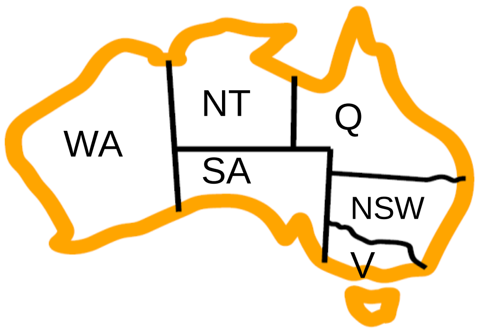
Which variable should be assigned next?{6.3.1}



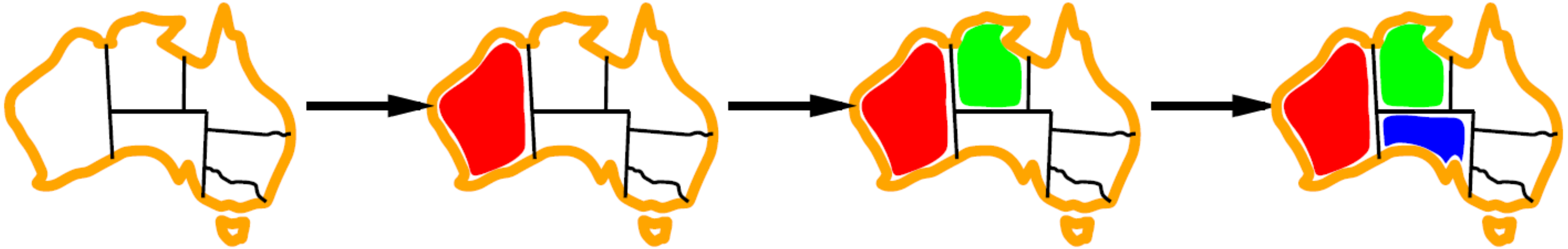


which variable?

Which variable should be assigned next?{6.3.1}



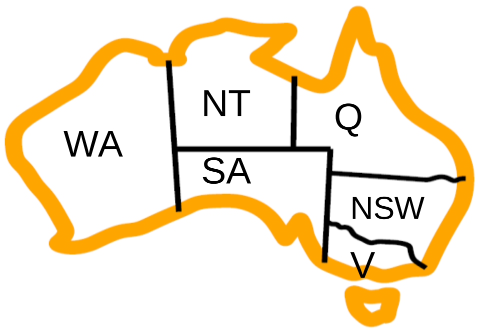
- Minimum remaining values (最少剩余值)
  - Choose the variable with the fewest legal values

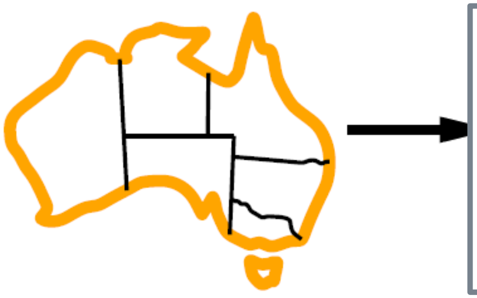


- also called "Fail-first" ordering (失败优先)

if the variables have the same number of remaining values, which variable first?

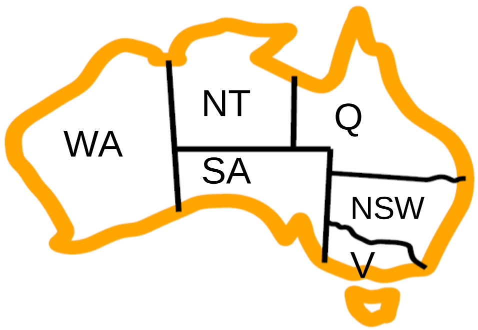
if the remaining variables have the same number of values {6.3.1}



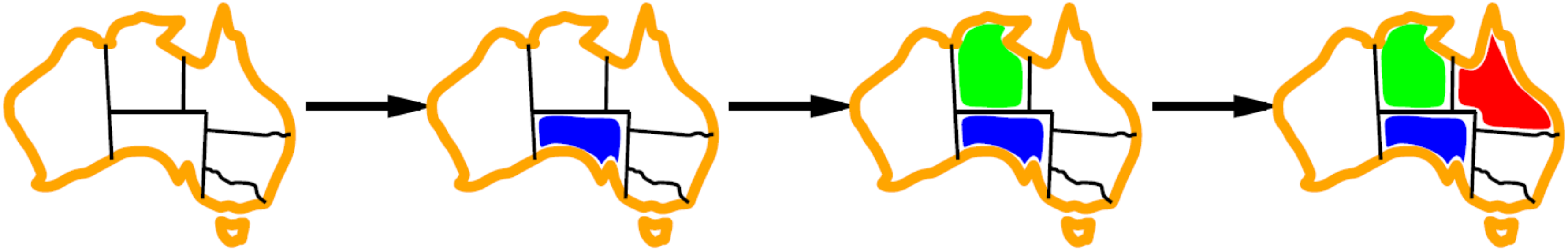


which variable?

if the remaining variables have the same number of values {6.3.1}

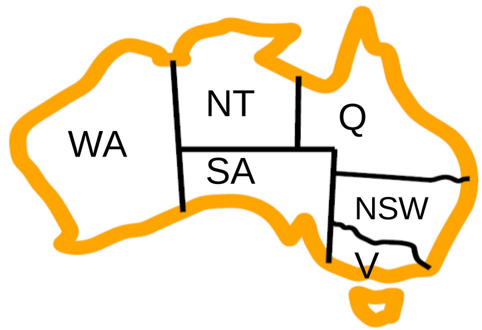


- Degree heuristic
  - Choose the variable with the most constraints on remaining variables

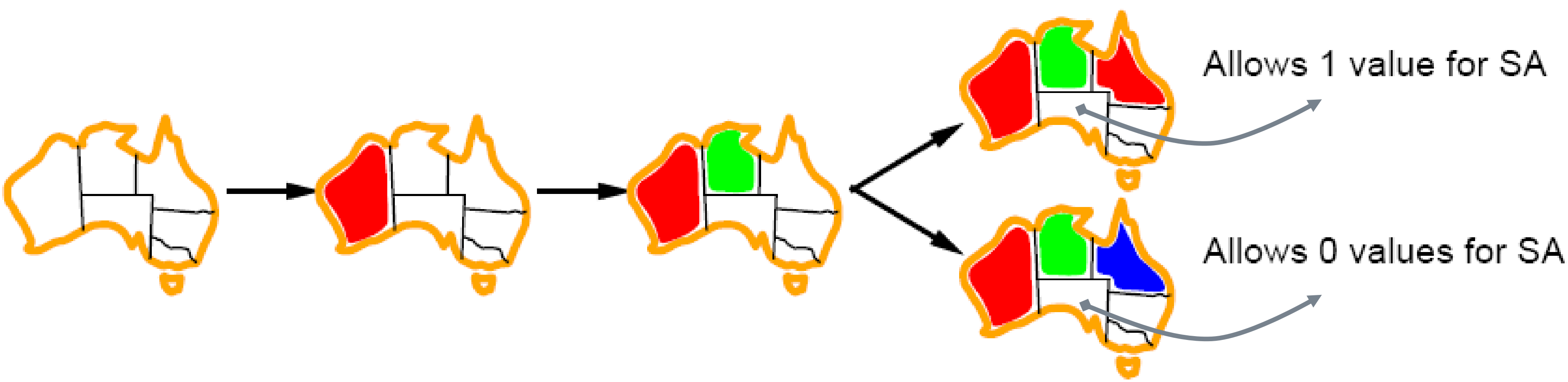


which value first,
once a variable is chosen?

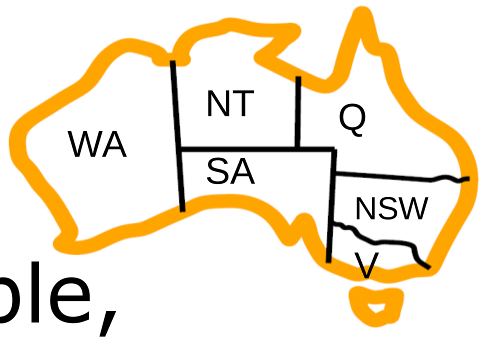
in what order should the variable's values be tried? {6.3.1}



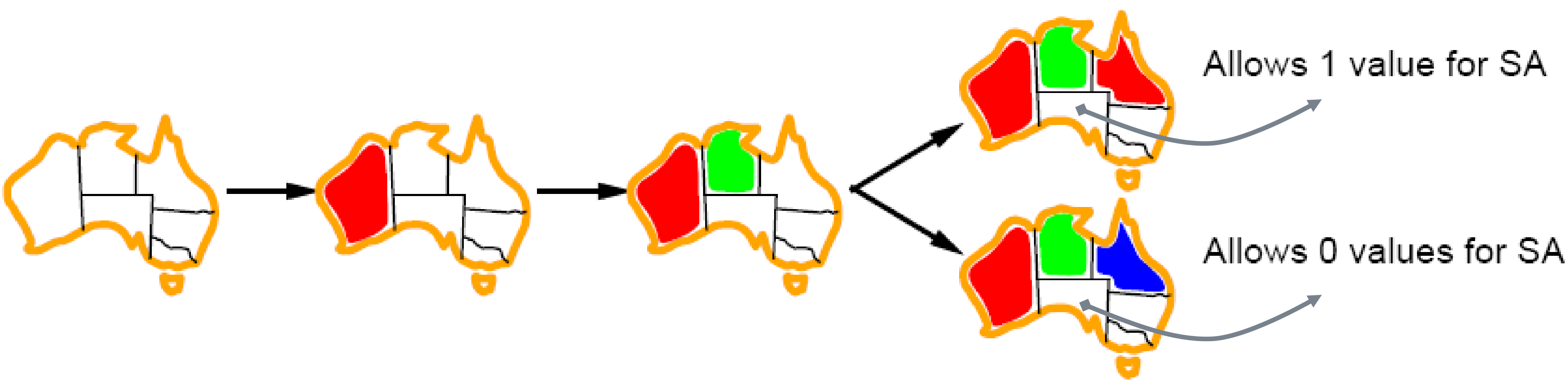
which value?



in what order should the variable's values be tried? {6.3.1}

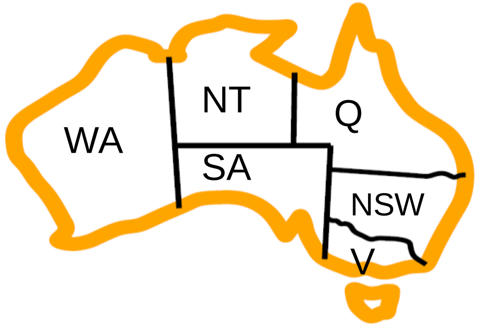


- Given a variable,
  choose the least constraining value
  - the one that rules out the fewest values for the neighboring variables



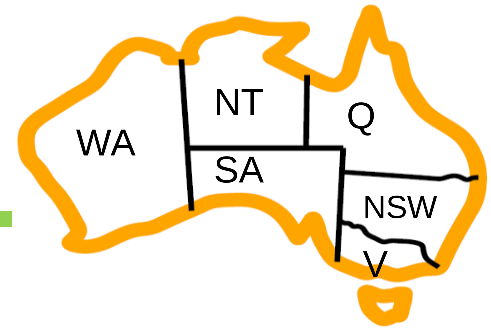
How to detect failure early?

How to detect inevitable failure early? {6.3.2}

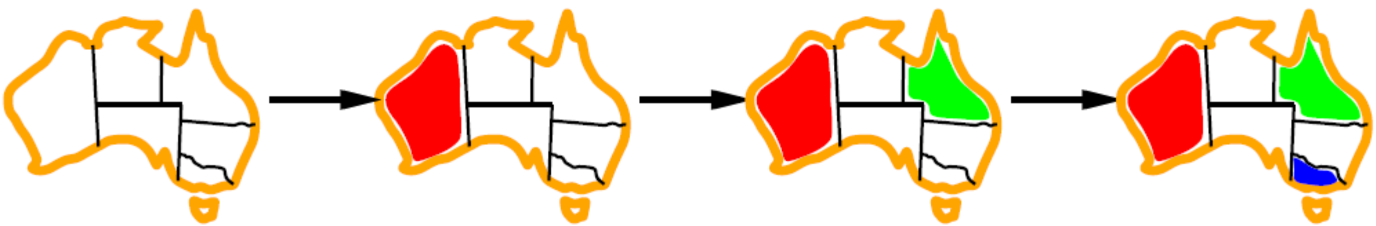


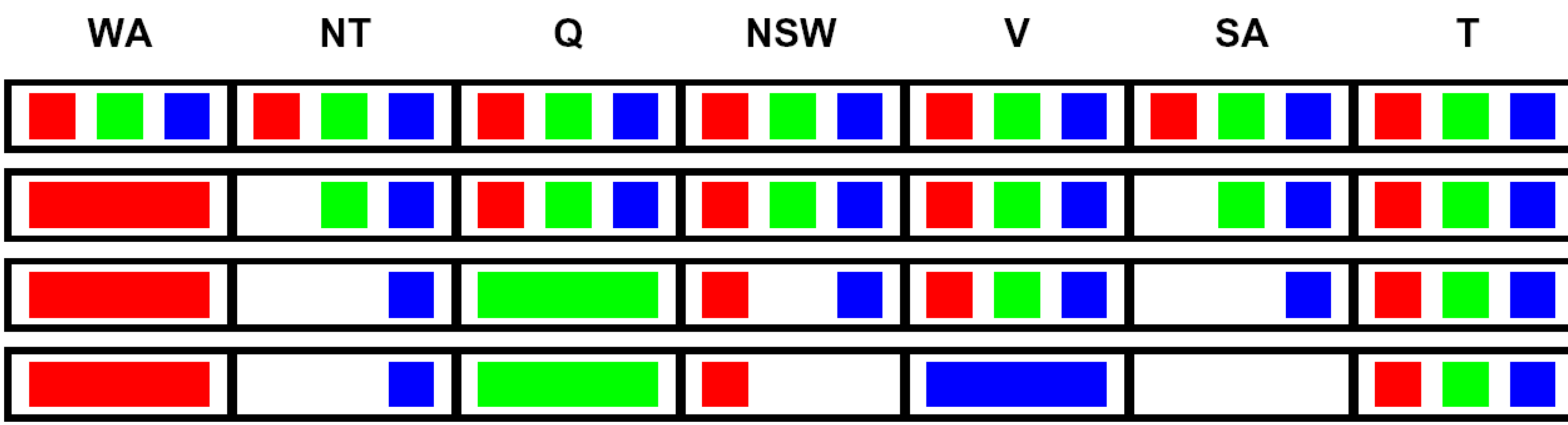
- Will it inevitably fail?

How to detect inevitable failure early? {6.3.2}



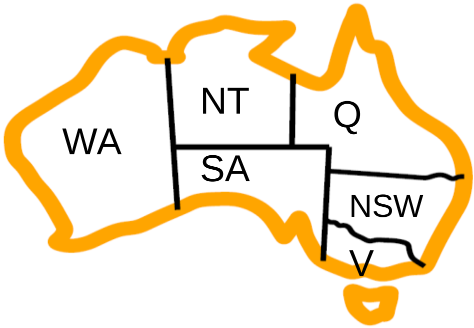
- Forward Checking: Keep track of remaining legal values for unassigned variables. Terminate when any variable has no legal values.



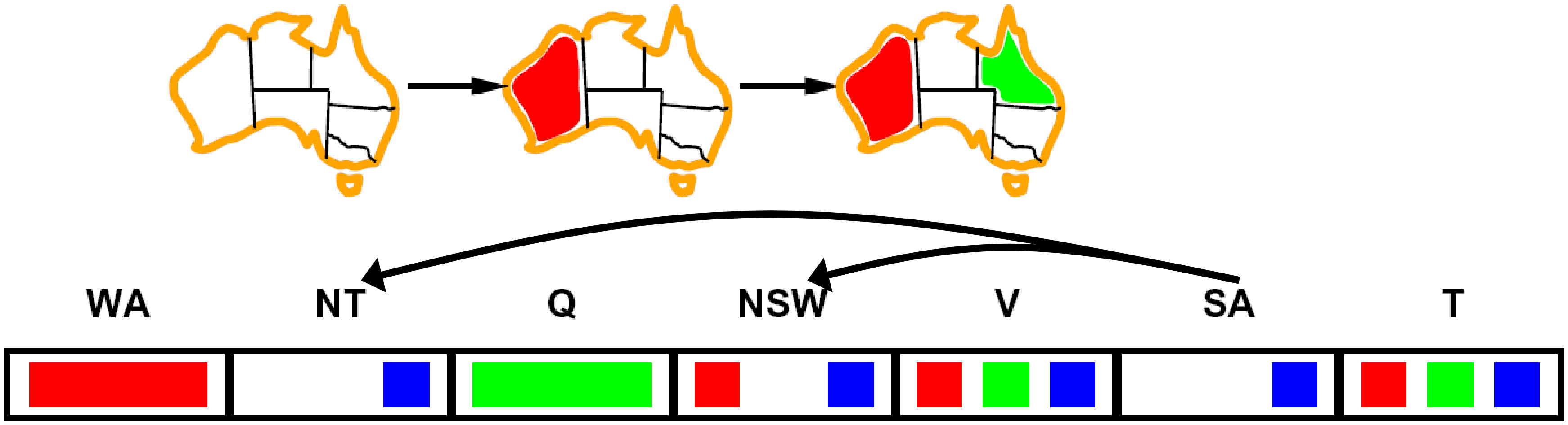


How to detect failure earlier than forward checking?

maintaining Arc Consistency {6.3.2}



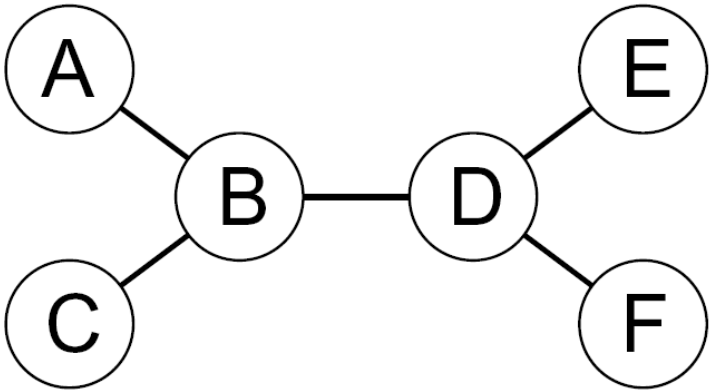
- Simplest form of propagation makes each arc consistent
- X $\rightarrow$ Y is consistent iff for every value x there is *some* allowed y



- Arc consistency detects failure earlier than forward checking
- Can be run as a preprocessor or after each assignment

Problem Structure

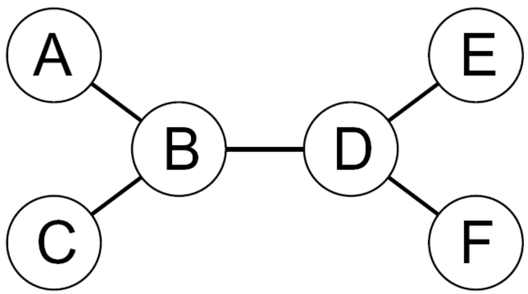
How to take advantage of problem structure? {6.5}

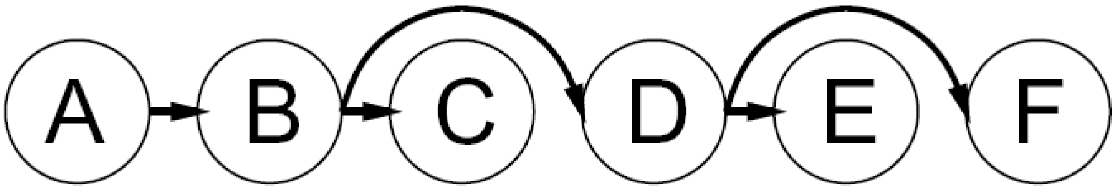


- Theorem: if the constraint graph has no loops, the CSP can be solved in $O(n\ d^2)$ time.
  HOW TO DO?
- Compare to general CSPs, where worst-case time is $O(d^n)$

Tree-Structured CSPs {6.5}

1. Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering





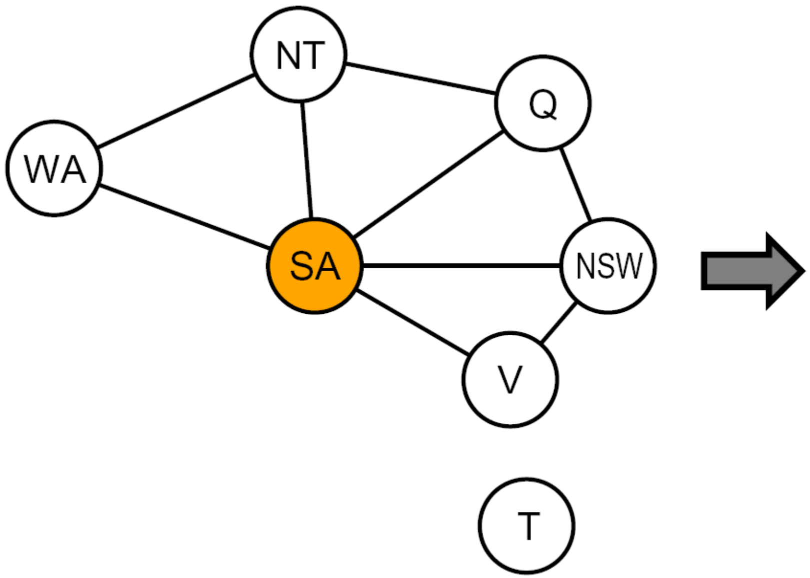
2. For i = n : 2, Make-Arc-Consistent(Parent(X_i),X_i)
3. For i = 1 : n, assign X_i consistently with Parent(X_i)

- Runtime: $O(n\ d^2)$

- Make_Arc_Consistent(X,Y)的功能是: 通过最少地删除X的值域中的值，使得X对于Y是弧相容的。

If the graph is not a tree:

Nearly Tree-Structured CSPs {6.5}



- Conditioning: instantiate a variable, prune its neighbors' domains
- Cutset conditioning: instantiate a set of variables such that the remaining constraint graph is a tree

(Finding the minimum cutset is NP-complete.)

Summary: backtracking {}

- Basic solution: DFS / backtracking / heuristic
- Forward checking:
  - Pre-filter unassigned domains after every assignment
  - Only remove values which conflict with current assignments
- Arc consistency
  - We only defined it for binary CSPs
  - Check for impossible values on all pairs of variables, prune them
  - Run (or not) after each assignment
  - A pre-filter, not search!

CSP AS A LOCAL SEARCH

local search for CSP

- 局部搜索算法（见第4.1节）对求解许多CSP都是很有效的。它们使用完整状态的形式化：初始状态是给每个变量都赋一个值，搜索过程是一次改变一个变量的取值。

CSP Summary {6.6}

- CSPs are a special kind of search problem:
  - States defined by values of a fixed set of variables
  - Goal test defined by constraints on variable values
- Backtracking = depth-first search with one legal variable assigned per node
- Variable ordering and value selection heuristics help significantly
- Forward checking prevents assignments that guarantee later failure
- Constraint propagation (e.g., arc consistency) does additional work to constrain values and detect inconsistencies
- problem structure
- local search can solve CSP

summary

1 何谓CSP：求解n-qeens、CSP形式化、约束图、变量与约束类型

2 约束传播与局部相容性(结点相容、弧相容、路径相容)

3 CSP形式化为一个搜索问题（回溯法）

4 提高搜索效率（变量顺序、值的顺序、提前检查失败、利用树形结构）

5 局部搜索可求解CSP

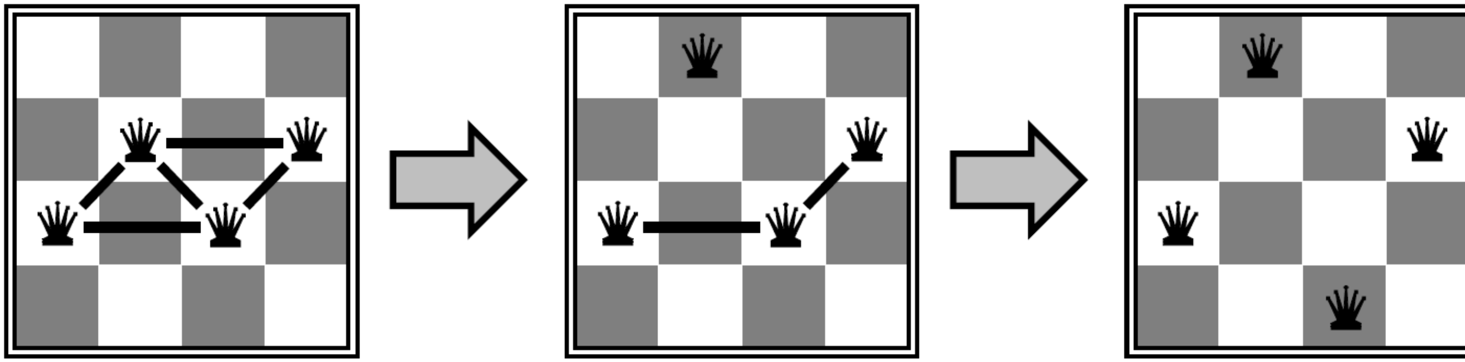
Exercise1: Varieties of Constraints

[5.11] Show how a single ternary constraint such as "A+B=C" can be turned into three binary constraints by using an auxiliary variable. You may assume finite domains. Next, show how constraints with more than 3 variables can be treated similarly. Finally, show how unary constraint can be eliminated. This completes the demonstration that any CSP can be transformed into a CSP with only binary constraint.

Exercise2: Street Puzzle

- **6.7** 考虑下述的逻辑问题：有5所不同颜色的房子，住着5个来自不同国家的人，每个人都喜欢一种不同牌子的糖果、不同牌子的饮料和不同的宠物。给定下列已知事实，请回答问题“斑马住在哪儿？哪所房子里的人喜欢喝水？
  1. 英国人(Englishman)住在红色(red)的房子里。
  2. 西班牙人(Spaniard)养狗(dog)。
  3. 挪威人(Norwegian)住在最左边的第一所房子里。
  4. 绿(green)房子是象牙色(ivory)房子的右边邻居。
  5. 喜欢抽Hershey牌巧克力的人住在养狐狸(fox)的人的旁边。
  6. 住在黄色(yellow)房子里的人喜欢Kit Kats糖果。
  7. 挪威人(Norwegian)住在蓝色(blue)房子旁边。
  8. 喜欢Smarties糖果的人养了一只蜗牛(snail)。
  9. 喜欢Snickers糖果的人喝桔汁(orange juice)。
  10. 乌克兰人(Ukrainian)喝茶(tea)。
  11. 日本人(Japanese)喜欢Milky Ways糖果。
  12. 喜欢Kit Kats糖果的人住在养马(horse)人的隔壁。
  13. 住在绿色(green)房子的人喜欢喝咖啡(Coffee)
  14. 住在中间房子里的人喜欢喝牛奶(milk)。
- 讨论把这个问题表示成CSP的不同方法。你认为哪种比较好，为什么？

4-Queens as a local search



h = 5　　h = 2　　h = 0

- States: 4 queens in 4 columns (4^4 = 256 states)
- Operators: move queen in column
- Goal test: no attacks
- Evaluation: h(n) = number of attacks

N-Queens as a CSP

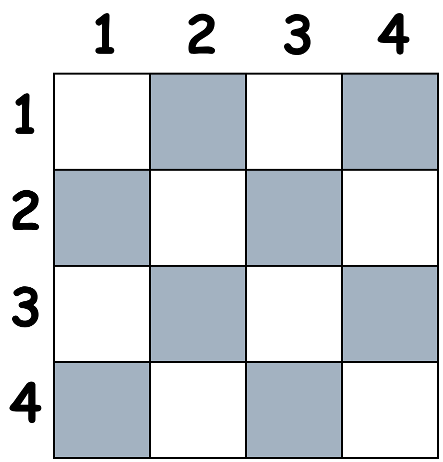
- Chessboard puzzle

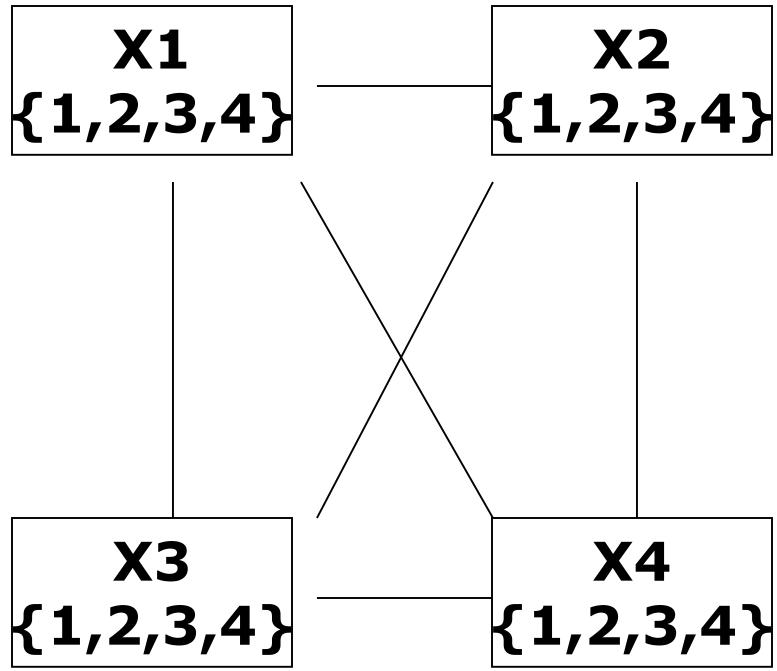
  e.g. when n = 8…

  - place 8 queens on a 8x8 chessboard so that no two attack each other
- Variable x_i for each column i of the board
- Domain = {1, 2, 3 … , n} for position in row
- Constraints are:
  - $x_i \neq x_j$ queens not in same row
  - $x_i - x_j \neq i-j$ queens not in same SE diagonal
  - $x_j - x_i \neq i - j$ queens not in SW diagonal

Example: 4-Queens Problem

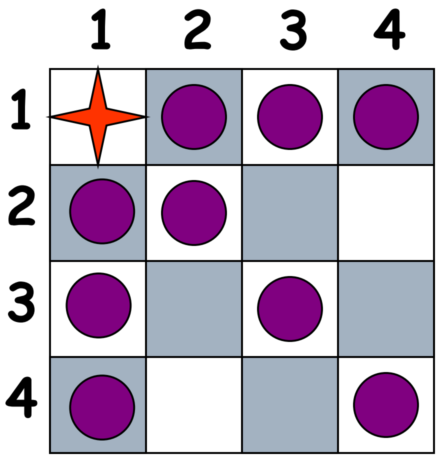
假设考虑变量的次序是**X1,X2,X3,X4**，考虑值的次序是从小到大考虑，采用前向检验。画出回溯搜索树，搜索过程发生几次回溯？

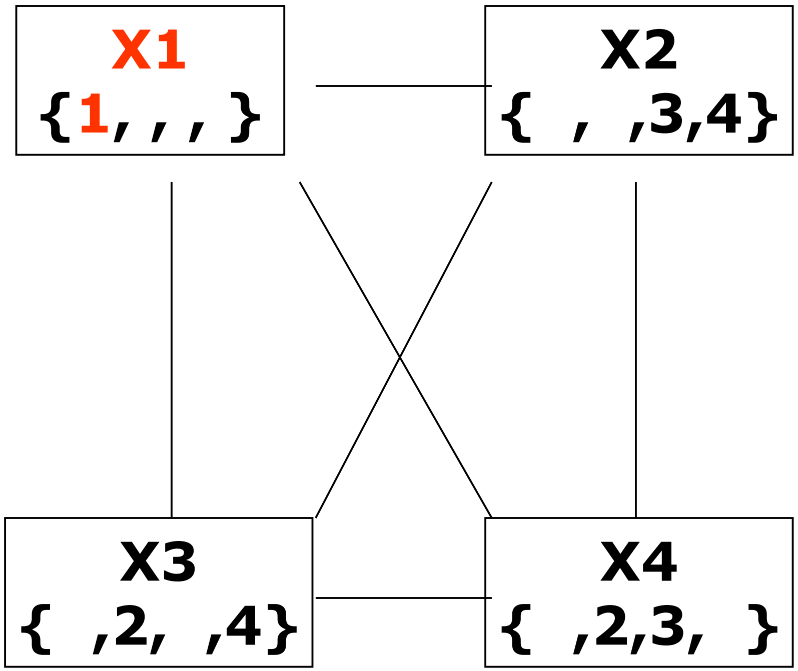




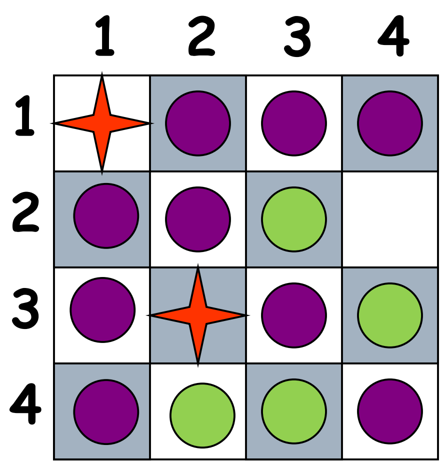
(From B.J. Dorr, U of Md, CMSC 421)

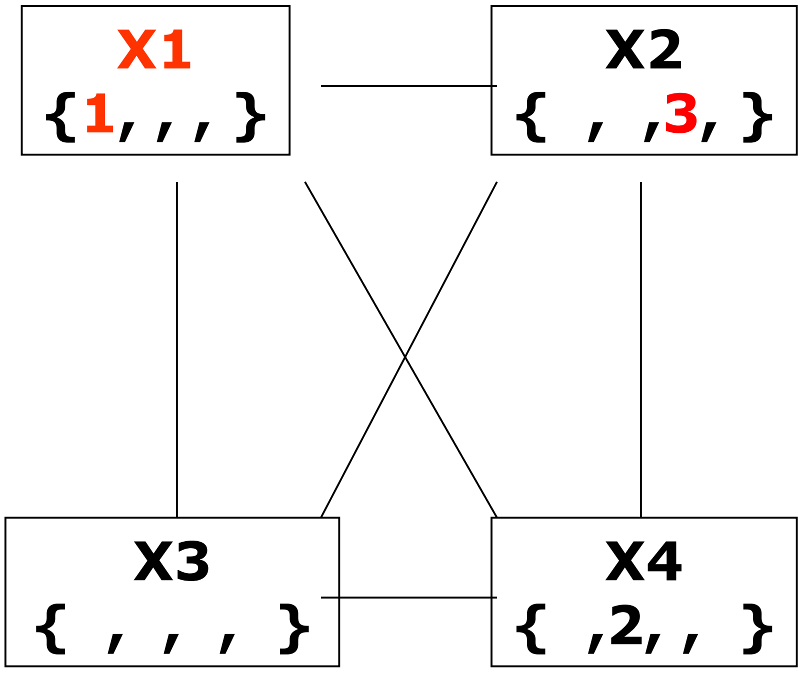
Example: 4-Queens Problem





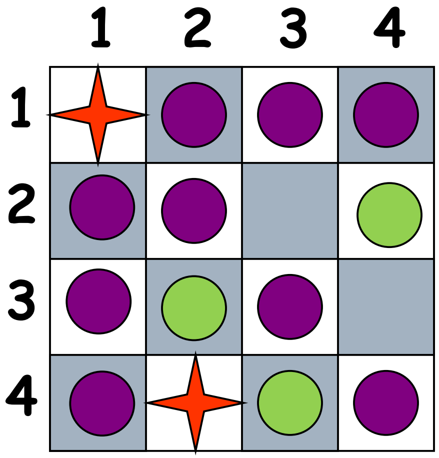
Example: 4-Queens Problem

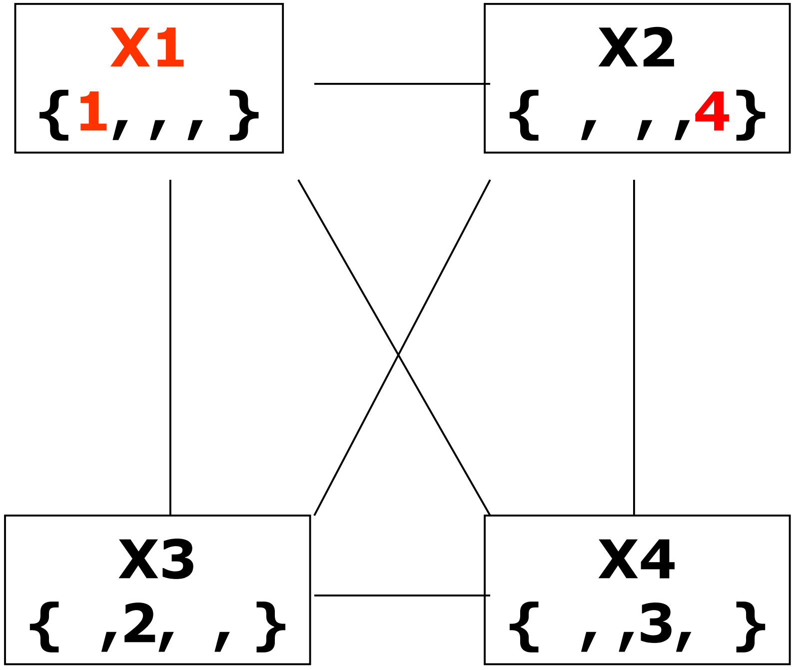




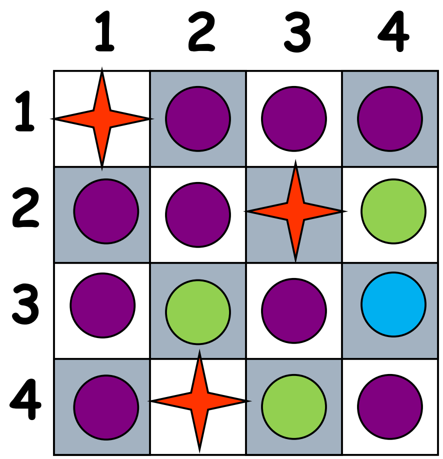
Backtrack!!!

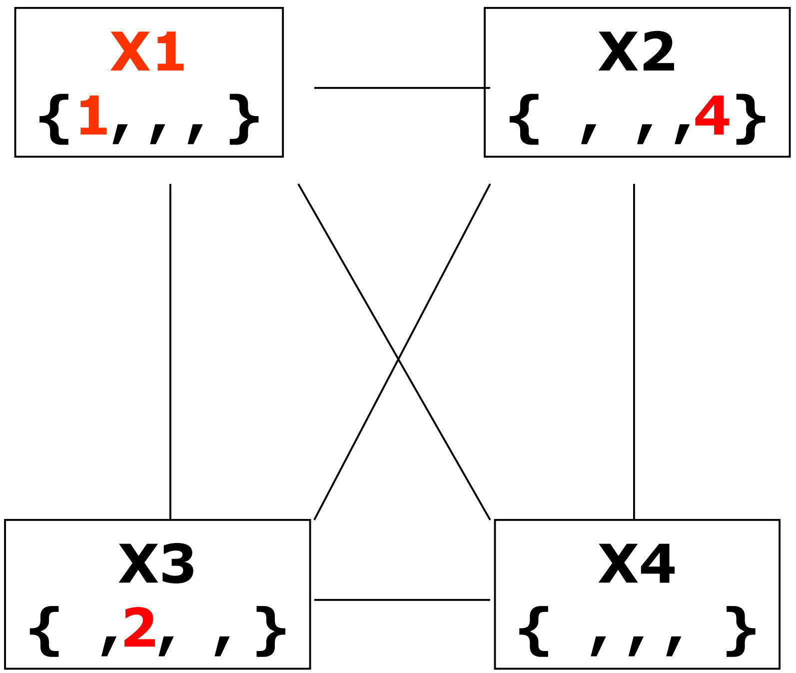
Example: 4-Queens Problem





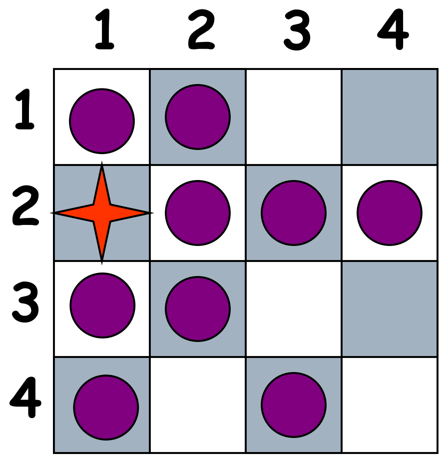
Example: 4-Queens Problem

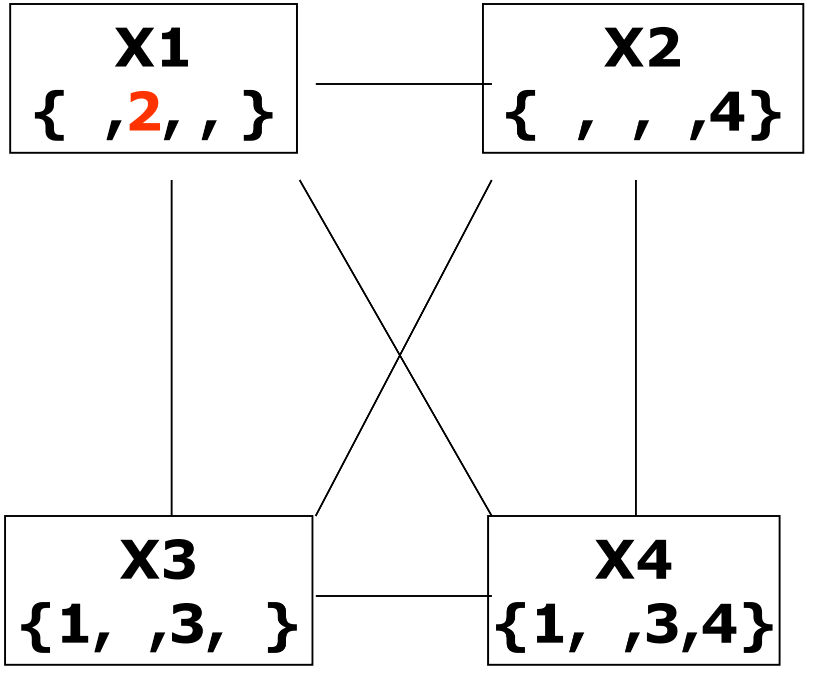




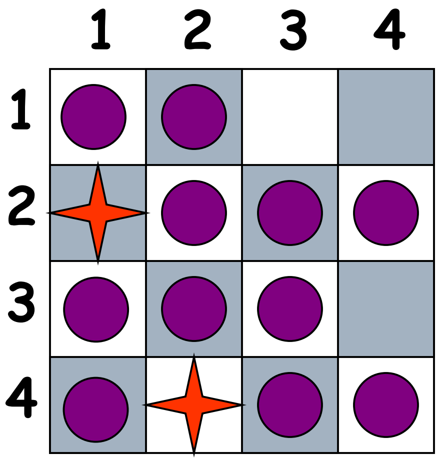
Backtrack!!!

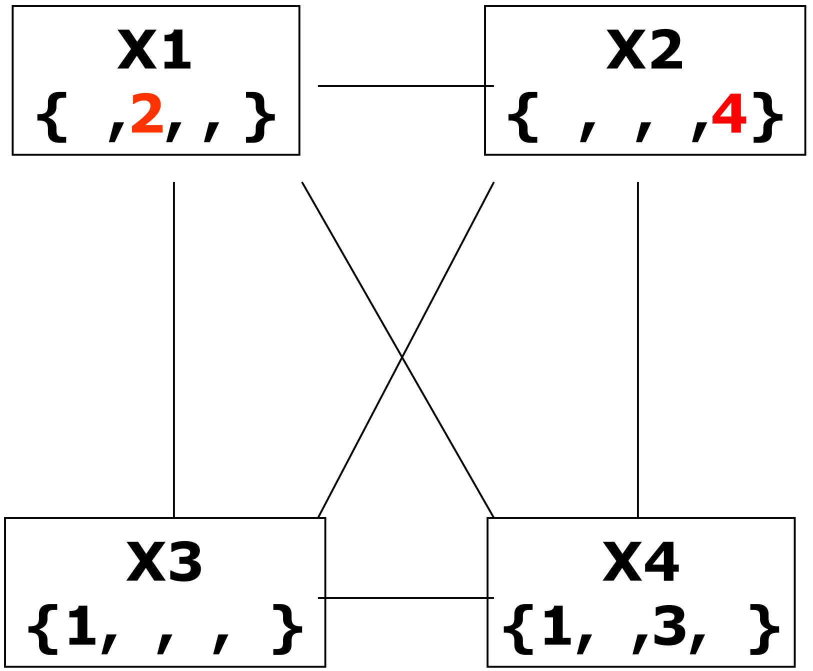
Example: 4-Queens Problem



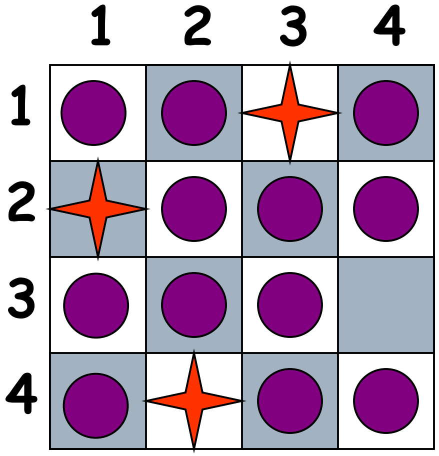


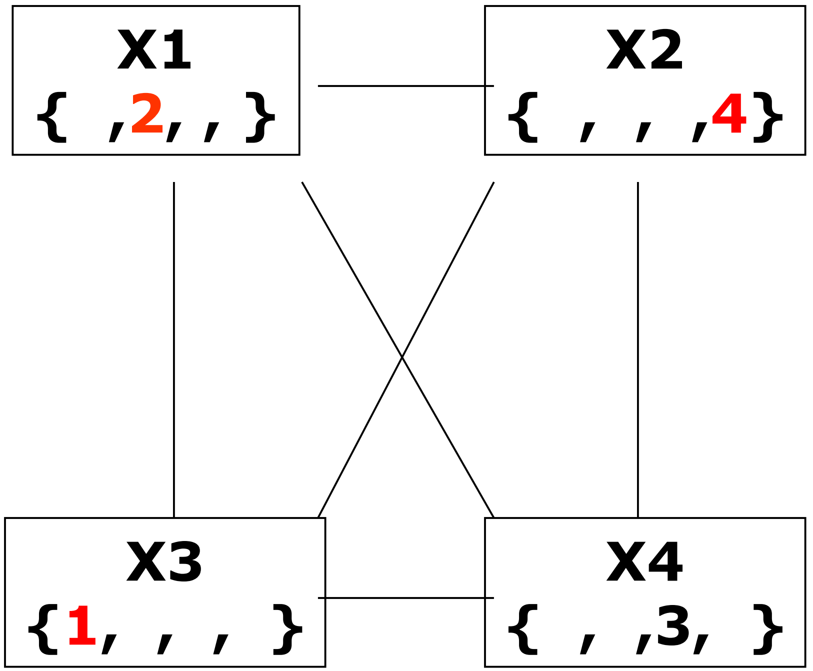
Example: 4-Queens Problem



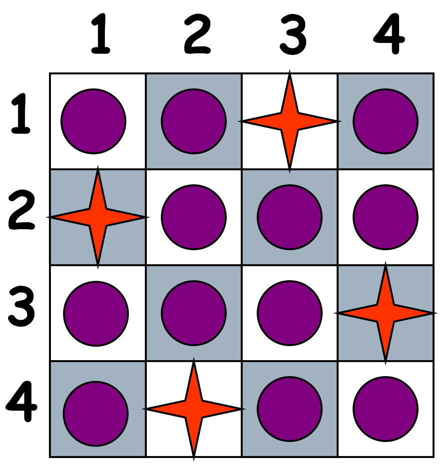


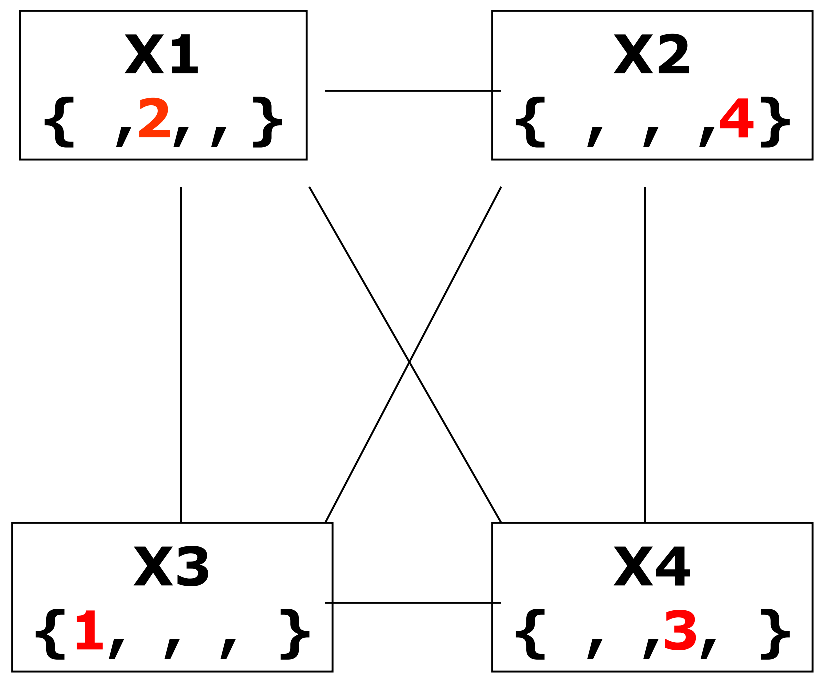
Example: 4-Queens Problem





Example: 4-Queens Problem





Boolean satisfiability problem (SAT)*

- Given a Boolean expression, is it satisfiable?
- Very basic problem in computer science

$$p_1 \wedge (p_2 \rightarrow p_3) \wedge ((\neg p_1 \wedge \neg p_3) \rightarrow \neg p_2) \wedge (p_1 \vee p_3)$$

- Turns out you can always express in 3-CNF

$$(p_1) \wedge (\neg p_2 \vee p_3) \wedge (p_1 \vee p_3 \vee \neg p_2) \wedge (p_1 \vee p_2 \vee p_3)$$

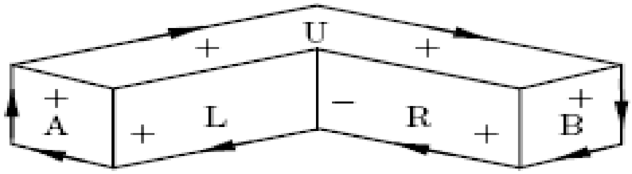
- 3-SAT: find a satisfying truth assignment

Example: 3-SAT*

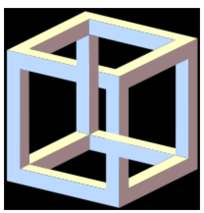
- Variables: $p_1, p_2, \ldots p_n$
- Domains: $\{\text{true}, \text{false}\}$
- Constraints:

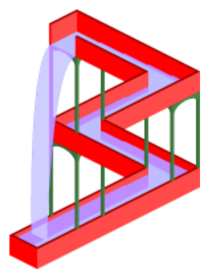
$$\left.\begin{array}{c} p_i \vee p_j \vee p_k \\ \neg p_{i'} \vee p_{j'} \vee p_{k'} \\ \vdots \\ p_{i''} \vee \neg p_{j''} \vee \neg p_{k''} \end{array}\right\}$$

Implicitly conjoined (all clauses must be satisfied)



next
chapter 7 logical agent





Thanks!