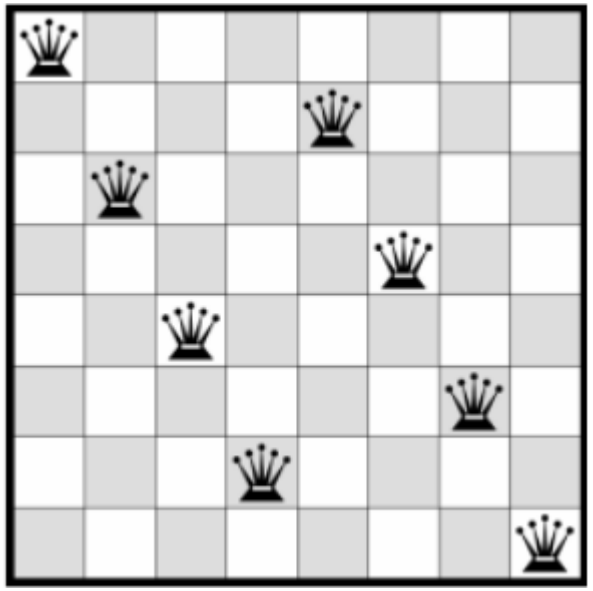
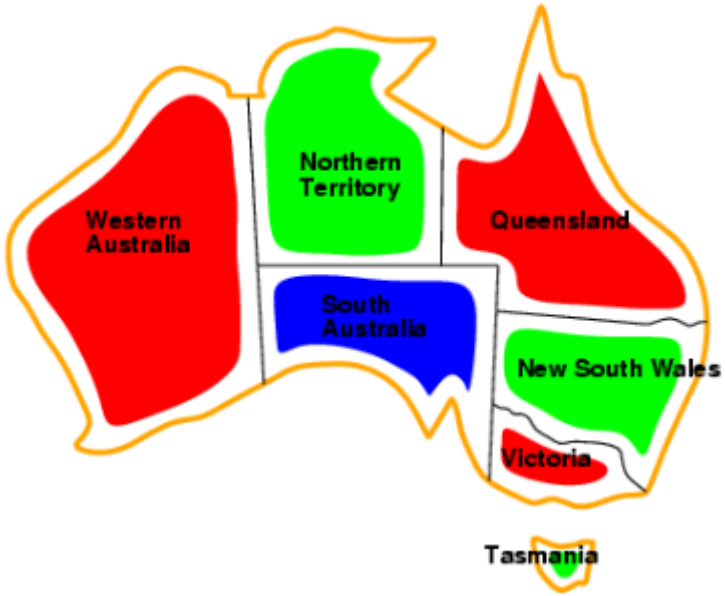
Constraint Satisfaction Problems
(ARTIFICIAL INTELLIGENCE)

(slides adapted and revised from Dan Klein, Pieter Abbeel, Anca Dragan, et al)

CÁC BÀI TOÁN

- **Bài toán 8 quân hậu**
- **Bài toán tô màu đồ thị**
- **Bài toán giải mã các ký tự**

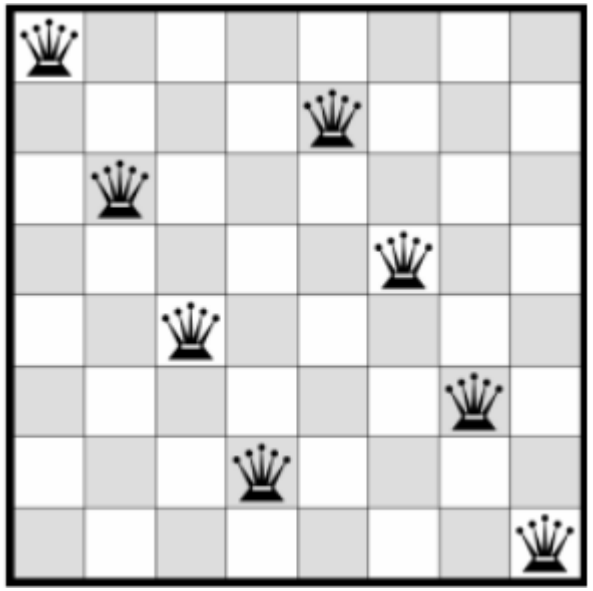





```
  T W O
+ T W O
-------
F O U R
```

Bài toán 8 quân hậu

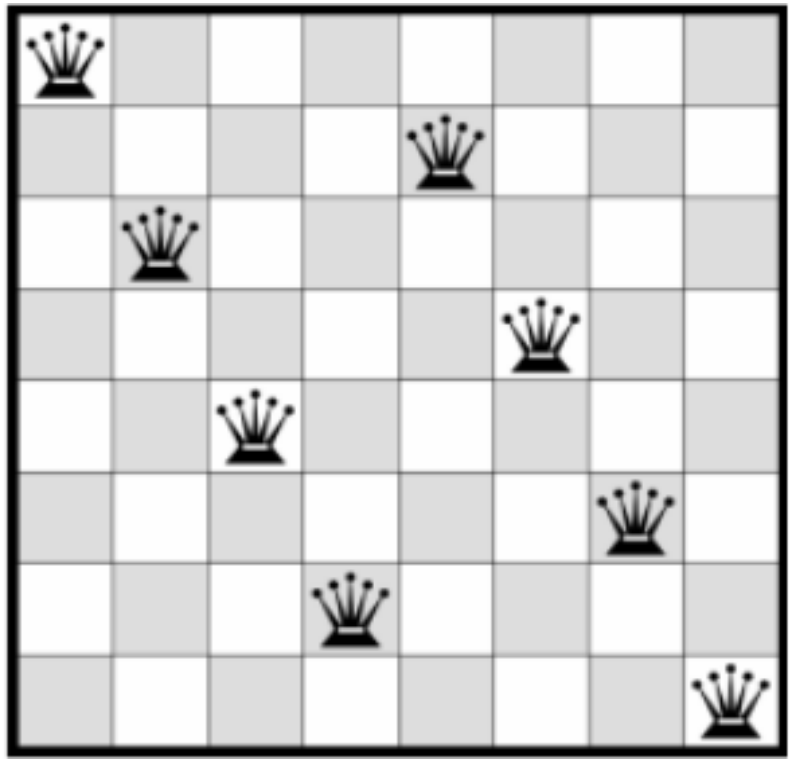
- **Phát biểu bài toán 8 quân hậu dưới dạng bài toán tìm kiếm có ràng buộc?**



Bài toán 8 quân hậu

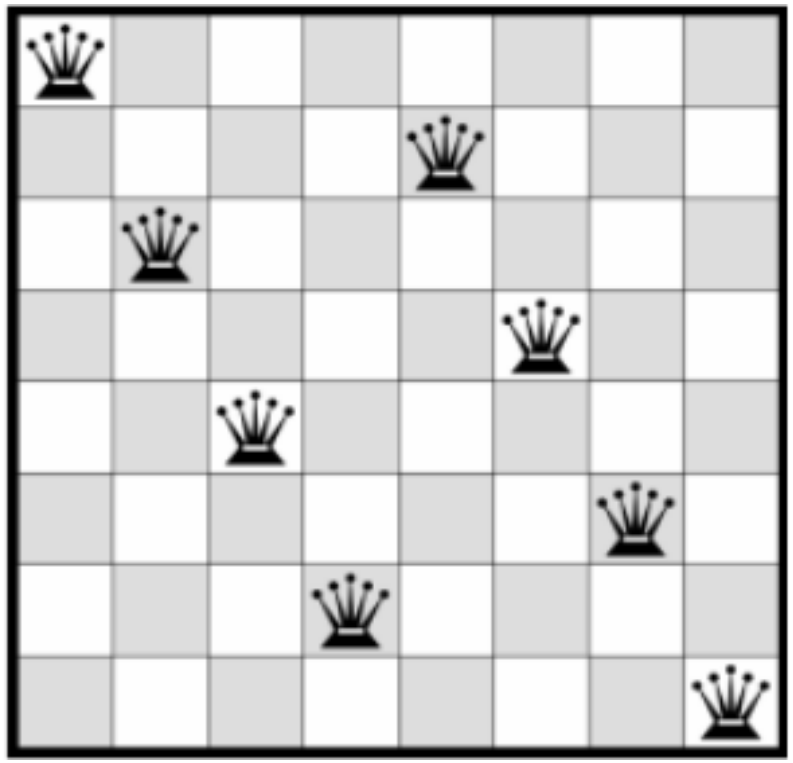
- **Phát biểu bài toán 8 quân hậu dưới dạng bài toán tìm kiếm có ràng buộc?**

=> Hãy đặt trên bàn cờ 8 quân hậu sao cho không có hai quân hậu nào cùng hang hoặc cùng cột hoặc cùng đường chéo.



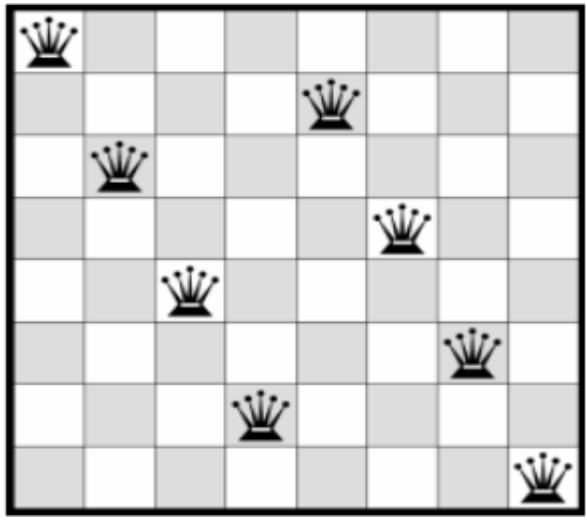
Bài toán 8 quân hậu

- **Bài toán 8 quân hậu có thể biểu diễn bởi 5 thành phần của bài toán tìm kiếm như thế nào?**



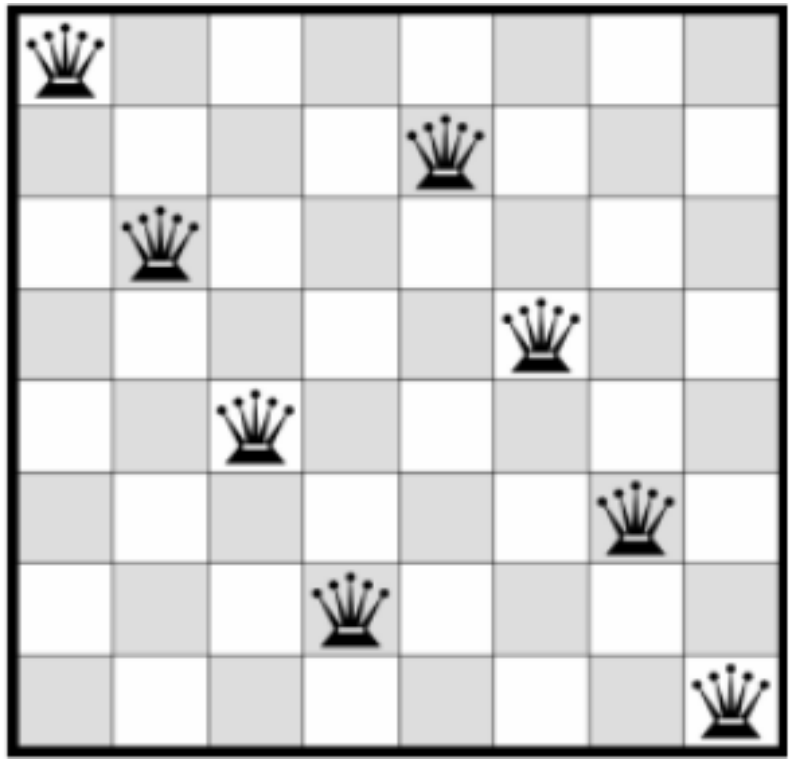
Bài toán 8 quân hậu

- **Bài toán 8 quân hậu có thể biểu diễn bởi 5 thành phần nào?**
- Không gian trạng thái?
  - mảng một chiều 8 phần tử HAU[0,1,…,7], phần tử HAU[i] biểu diễn dòng đặt con hậu cột i. Ví dụ HAU[i]=j có nghĩa là con hậu cột i đặt ở dòng j.
- Các hành động?
  - thêm một quân hậu vào bất kỳ ô trống nào
- Trạng thái xuất phát?
  - Một mảng ngẫu nhiên 8 phần tử, mỗi phần tử nhận giá trị từ 0 đến 7
- Trạng thái mục tiêu?
  - Gán các giá trị khác nhau phạm vi từ 0 đến 7 cho các phần tử của mảng sao cho i-HAU[i] ≠ j-HAU[j] (không nằm trên cùng đường chéo phụ) và i+HAU[i] ≠ j + HAU[j] (không nằm trên cùng đường chéo chính).
- Chi phí?
  - Không xác định



Bài toán 8 quân hậu

- **Để biểu diễn bài toán 8 quân hậu dưới dạng bài toán tìm kiếm thỏa mãn ràng buộc, cần những thành phần nào?**

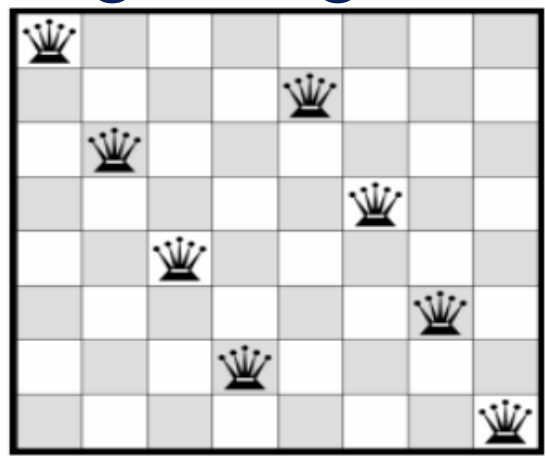


Bài toán 8 quân hậu

- **Để biểu diễn bài toán 8 quân hậu dưới dạng bài toán tìm kiếm thỏa mãn ràng buộc, cần những thành phần nào?**

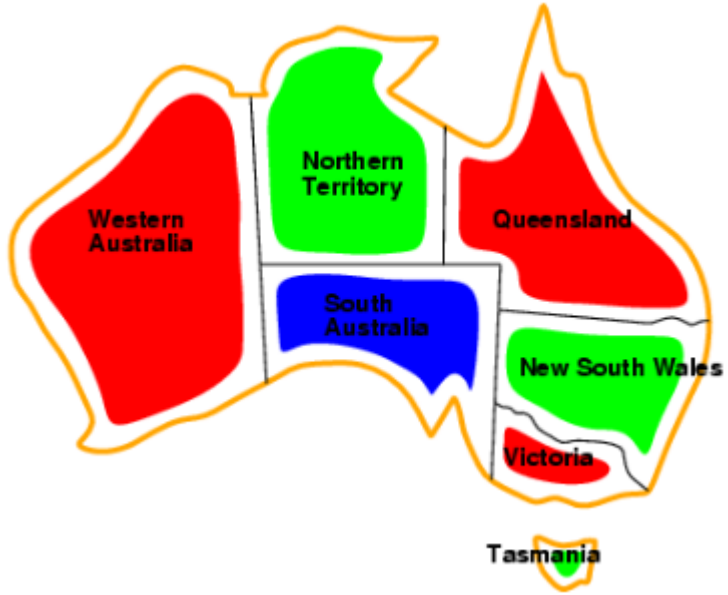
- Tập các biến mô tả trạng thái của bài toán: HAU[0], HAU[1], .., HAU[7] trong bài toán 8 quân hậu (HAU[i] là số hiệu dòng đặt con hậu ở cột I, ví dụ HAU[0]=0 có nghĩa là con hậu cột đầu tiên (cột 0) sẽ đặt ở dòng đầu tiên (dòng 0).
- Miền giá trị cho các biến: HAU[i] $\in$ {0, 1, 2, 3, 4, 5, 6, 7}
- Tập ràng buộc: với i≠j thì HAU[i] ≠HAU[j] (không có hai con hậu cùng hàng ngang), i-HAU[i] ≠ j-HAU[j] (không có hai con hậu nào cùng đường chéo phụ); i+HAU[i] ≠ j+HAU[j] (không có hai con hậu nào cùng đường chéo chính)

=> Lời giải của bài toán là một phép gán giá trị trong miền giá trị cho các biến sao cho thỏa mãn các ràng buộc của bài toán.



Bài toán tô màu đồ thị

- **Phát biểu bài toán tô màu đồ thị dưới dạng bài toán tìm kiếm có ràng buộc?**

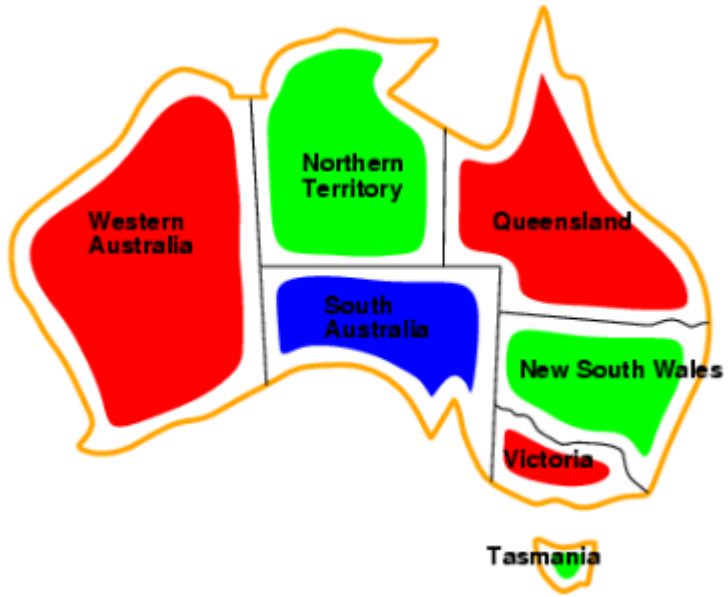


Bài toán tô màu đồ thị

- **Phát biểu bài toán tô màu đồ thị dưới dạng bài toán tìm kiếm có ràng buộc?**

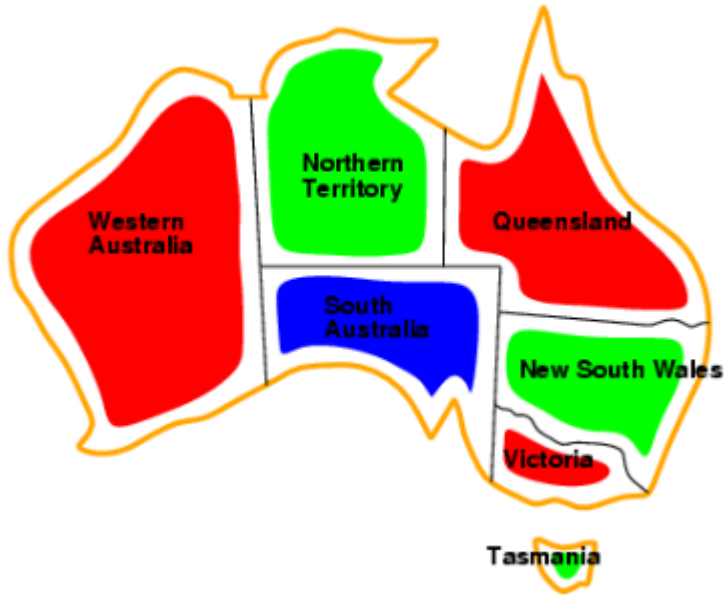
⇒Sử dụng ba màu để tô bản đồ các tỉnh của một nước sao cho các tỉnh kề nhau thì có màu khác nhau.

⇒Ví dụ, nước Australia có 7 bang như hình vẽ, chỉ sử dụng ba màu: đỏ, xanh lơ và xanh da trời để tô màu 7 bang của nước Australia sao cho không có hai bang nào kề nhau lại có màu giống nhau.



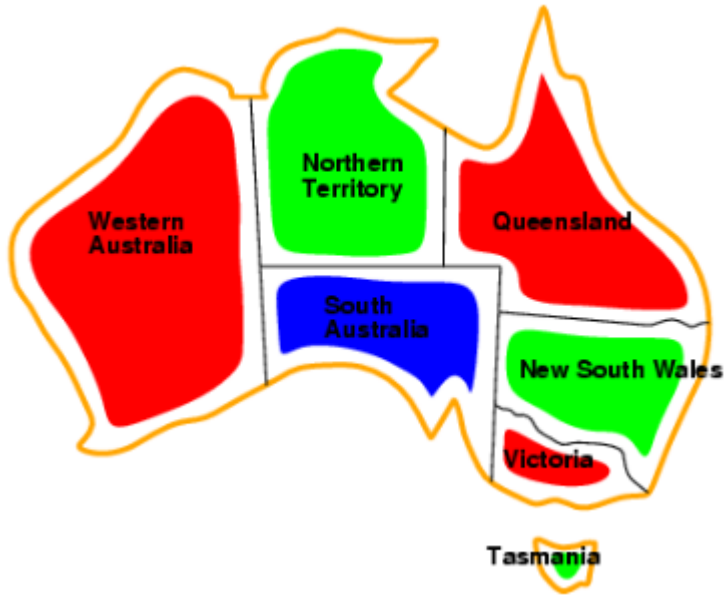
Bài toán tô màu đồ thị

- **Để biểu diễn bài toán tô màu đồ thị dưới dạng bài toán tìm kiếm thỏa mãn ràng buộc, cần những thành phần nào?**



Bài toán tô màu đồ thị

- **Để biểu diễn bài toán tô màu đồ thị dưới dạng bài toán tìm kiếm thỏa mãn ràng buộc, cần những thành phần nào?**



⇒ Tập các biến: WA, NT, Q, NSW, V, SA, T (các biến là các ký tự đầu của tên các bang)

⇒ Miền giá trị: 7 biến có thể nhận các giá trị trong tập {đỏ, xanh lá cây, xanh da trời}

⇒Tập ràng buộc: WA≠NT, WA≠SA, NT≠SA, NT≠Q, SA≠Q, SA≠NSW, SA≠V, Q≠NSW, NSW≠V

⇒Lời giải của bài toán tô màu đồ thị là phép gán các giá trị {đỏ, xanh da trời, xanh lá cây} cho tập 7 biến thỏa mãn tập các ràng buộc.

Bài toán giải mã các ký tự

- **Phát biểu bài toán giải mã các ký tự dưới dạng bài toán tìm kiếm có ràng buộc?**

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

Bài toán giải mã các ký tự

- **Phát biểu bài toán giải mã các ký tự dưới dạng bài toán tìm kiếm có ràng buộc?**

⇒Tìm các chữ số thích hợp cho các ký tự để phép tính sau là đúng:

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

Bài toán giải mã các ký tự

- **Để biểu diễn bài toán giải mã các ký tự dưới dạng bài toán tìm kiếm thỏa mãn ràng buộc, cần những thành phần nào?**

$\Rightarrow$ Tập các biến: T, W, O, F, U, R, N1, N2, N3 (N1, N2, N3 là 3 số nhớ của phép cộng ở các vị trí hàng đơn vị, hàng chục, hàng trăm)

$\Rightarrow$ Miền giá trị: Các biến có thể nhận các giá trị: {0, 1, .., 9}

$\Rightarrow$Ràng buộc: T, W, O, F, U, R phải khác nhau đôi một;

$\Rightarrow$ O + O = R +10.N1;

$\Rightarrow$ N1 + W +W = U + 10.N2;

$\Rightarrow$ N2 + T + T = O + 10.N3;

$\Rightarrow$ F=N3; T≠0; F≠0

$\Rightarrow$ Lời giải của bài toán là một phép gán các chữ số từ 0 đến 9 cho các biến và thỏa mãn tập các ràng buộc

Giải bài toán tìm kiếm thỏa mãn ràng buộc

- Việc giải bài toán thỏa mãn các ràng buộc là tìm ra một phép gán giá trị cho tập các biến của bài toán sao cho tập các ràng buộc được thỏa mãn.
- Các bài toán lớp này được giải bằng giải thuật quay lui vét cạn

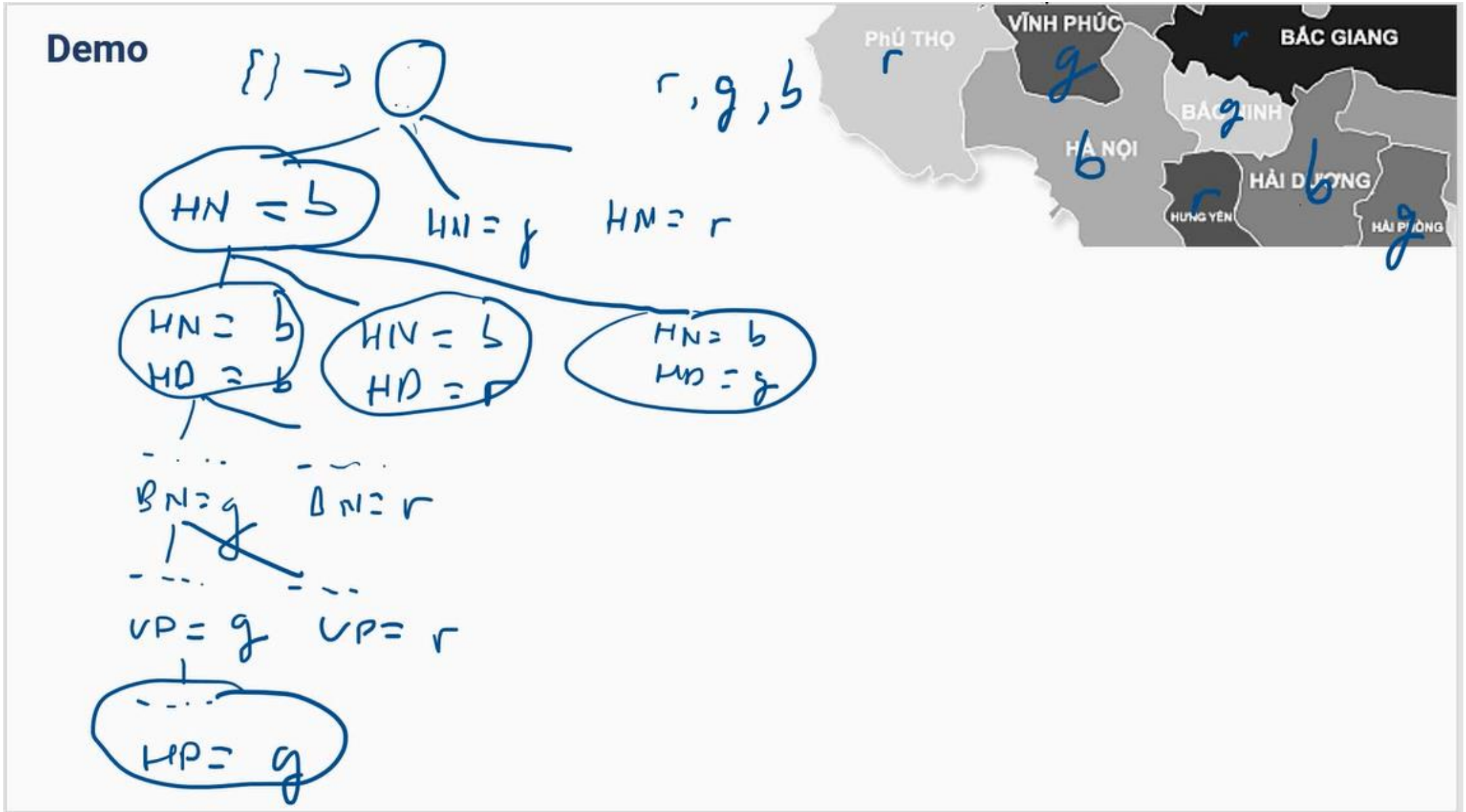
Giải thuật quay lui vét cạn

- Giả sử bài toán cần gán giá trị cho n biến, chúng ta có thể tìm lời giải của bài toán bằng các bước mô tả như sau:
  - Chọn 1 biến chưa được điền gía trị
  - Gán 1 giá trị theo miền giá trị cho phép của nó
  - Lặp lại bước 1,2 cho tới khi:
    - Tìm ra solution
    - Gặp một empty domain cho 1 biến nào đó

Giải thuật quay lui vét cạn

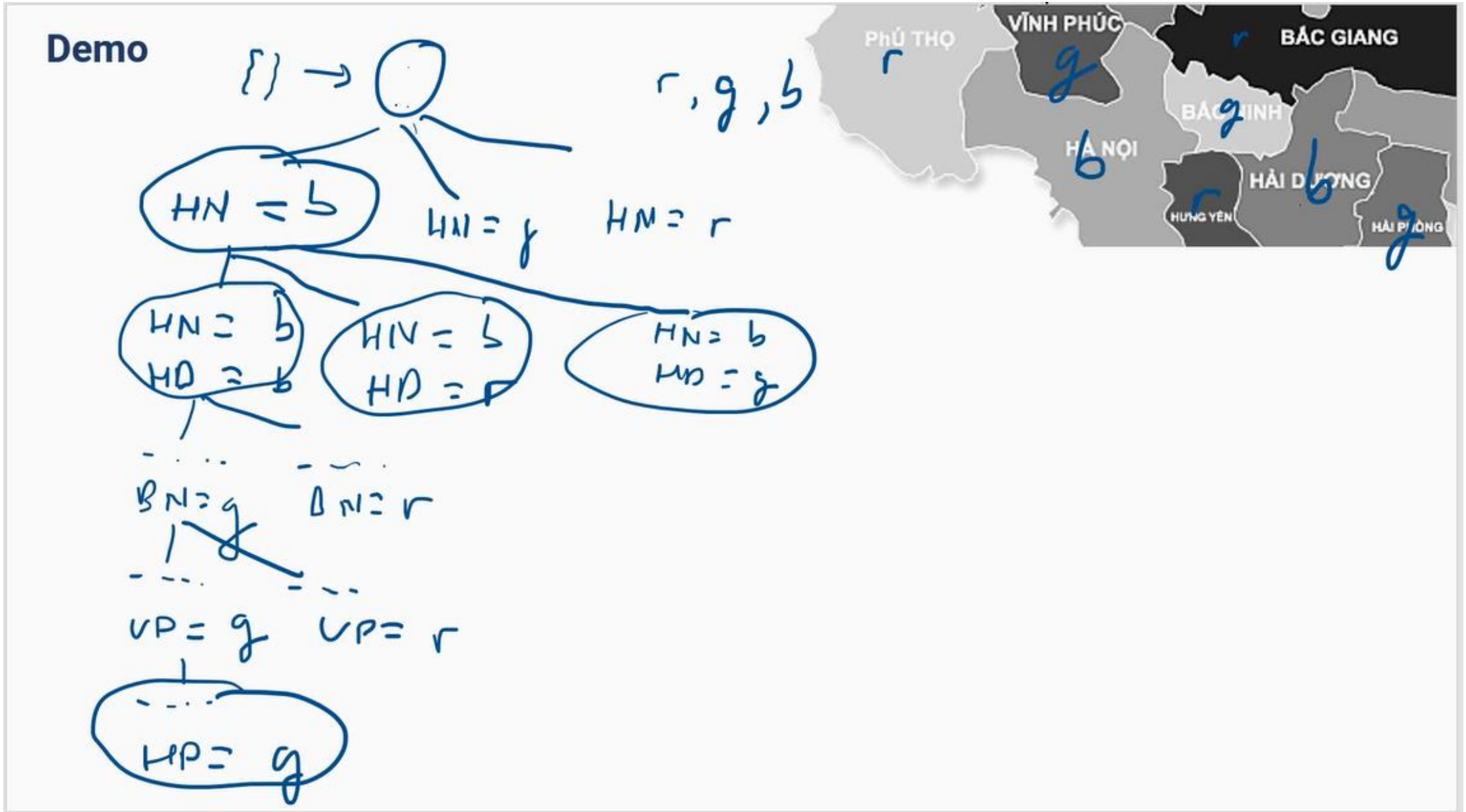
```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
    return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment then
            add {var = value} to assignment
            inferences ← INFERENCE(csp, var, value)
            if inferences ≠ failure then
                add inferences to assignment
                result ← BACKTRACK(assignment, csp)
                if result ≠ failure then
                    return result
        remove {var = value} and inferences from assignment
    return failure
```

Giải thuật quay lui vét cạn

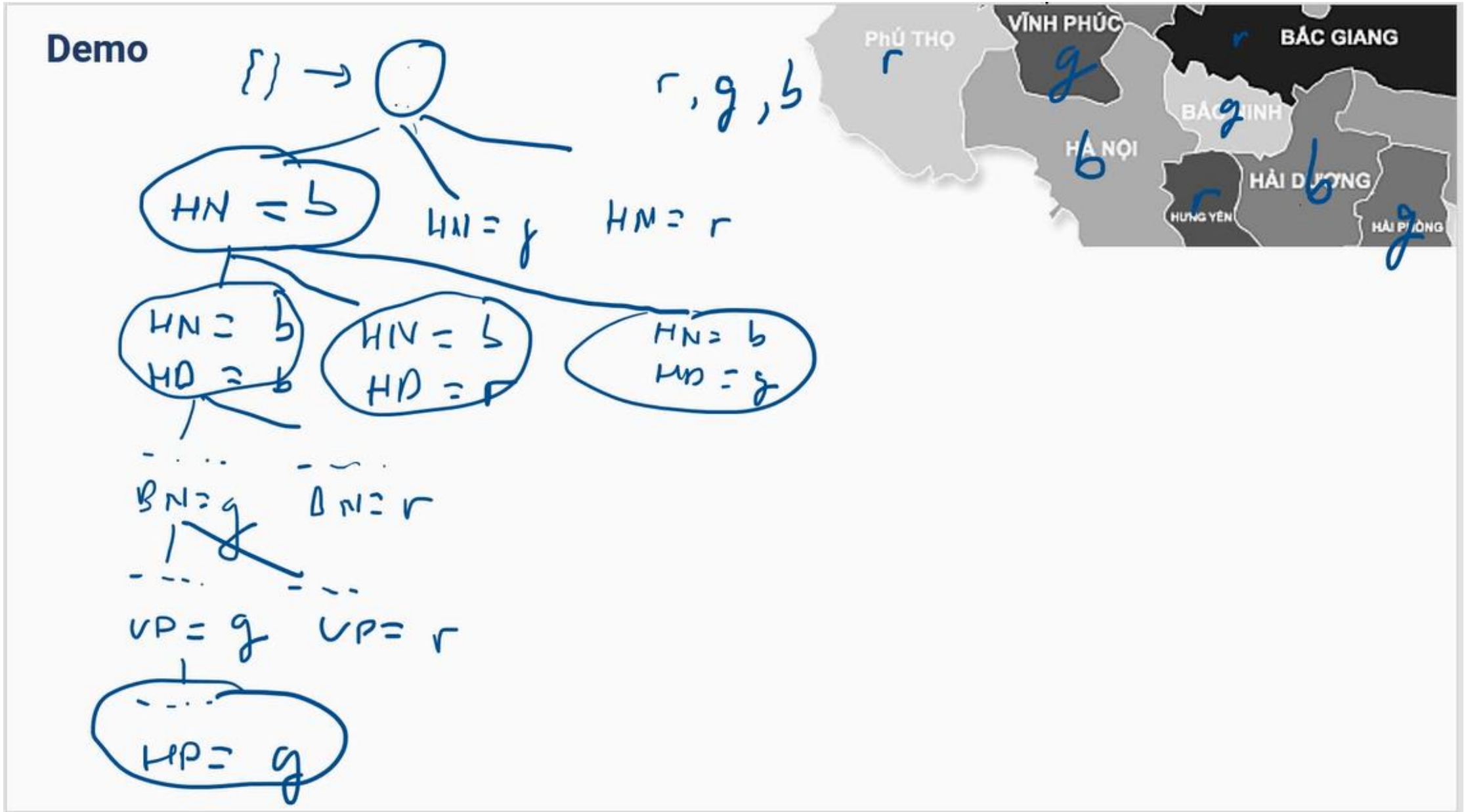



PHÚ THỌ
VĨNH PHÚC
BẮC GIANG
BẮC NINH
HÀ NỘI
HẢI DƯƠNG
HƯNG YÊN
HẢI PHÒNG

Giải thuật quay lui vét cạn



Giải thuật quay lui vét cạn



Cách giải thuật hoạt động

- Thuật toán backtracking hoạt động dựa trên việc thử từng lựa chọn có thể và quay lui khi gặp điều kiện không thỏa mãn.
  - Bước 1 khởi tạo.
  - Bước 2 kiểm tra điều kiện.
  - Bước 3 lựa chọn và kiểm tra ràng buộc.
  - Bước 4 cập nhật trạng thái.
  - Bước 5 đệ quy.
  - Bước 6 quay lui.
  - Bước 7 trả về kết quả.

Ví dụ khi giải quyết bài toán Xếp hậu.

- **Bước 1**. Khởi tạo:
  - Tạo một bàn cờ rỗng kích thước NxN và đánh dấu tất cả các ô trống là chưa đặt quân hậu.

Ví dụ khi giải quyết bài toán Xếp hậu.

- **Bước 1**. Khởi tạo:
- **Bước 2**. Bắt đầu từ hàng đầu tiên:

Ví dụ khi giải quyết bài toán Xếp hậu.

- **Bước 1**. Khởi tạo:
- **Bước 2**. Bắt đầu từ hàng đầu tiên:
- **Bước 3**. Đệ quy:

Ví dụ khi giải quyết bài toán Xếp hậu.

- **Bước 1**. Khởi tạo:
- **Bước 2**. Bắt đầu từ hàng đầu tiên:
- **Bước 3**. Đệ quy:
  - Với mỗi ô trống trên hàng đó thử đặt quân hậu vào ô đó và kiểm tra xem nó có bị tấn công bởi quân hậu nào đã đặt trên bàn cờ hay không.
  - Nếu ô không bị tấn công, đánh dấu ô đó là đã đặt quân hậu và tiếp tục đệ quy để đặt quân hậu trên hàng tiếp theo.
  - Nếu ô bị tấn công, thử ô tiếp theo trên hàng đó.

Ví dụ khi giải quyết bài toán Xếp hậu.

- **Bước 1**. Khởi tạo:
- **Bước 2**. Bắt đầu từ hàng đầu tiên:
- **Bước 3**. Đệ quy:
- **Bước 4**. Quay lui (Backtrack):
  - Nếu không tìm được ô trống trên hàng đó để đặt quân hậu, quay lại hàng trước đó và thử đặt quân hậu vào ô tiếp theo trên hàng đó.

Ví dụ khi giải quyết bài toán Xếp hậu.

- **Bước 1**. Khởi tạo:
- **Bước 2**. Bắt đầu từ hàng đầu tiên:
- **Bước 3**. Đệ quy:
- **Bước 4**. Quay lui (Backtrack):
- **Bước 5**. Lưu trữ kết quả:
  - Nếu đã đặt được n quân hậu trên bàn cờ, lưu trữ kết quả và thoát khỏi đệ quy.

Ví dụ khi giải quyết bài toán Xếp hậu.

- **Bước 1**. Khởi tạo:
- **Bước 2**. Bắt đầu từ hàng đầu tiên:
- **Bước 3**. Đệ quy:
- **Bước 4**. Quay lui (Backtrack):
- **Bước 5**. Lưu trữ kết quả:
- **Bước 6**. Quay lại và thử cách đặt quân hậu khác:
  - Nếu đã kiểm tra tất cả các ô trên bàn cờ nhưng không tìm được cách đặt quân hậu hợp lệ, quay lại và thử cách đặt quân hậu khác.
  - Quy trình này sử dụng đệ quy để thử từng ô trống trên mỗi hàng của bàn cờ và sử dụng quay lui để thử lại khi không thể đặt quân hậu trong một số trường hợp.

Ví dụ khi giải quyết bài toán Xếp hậu.

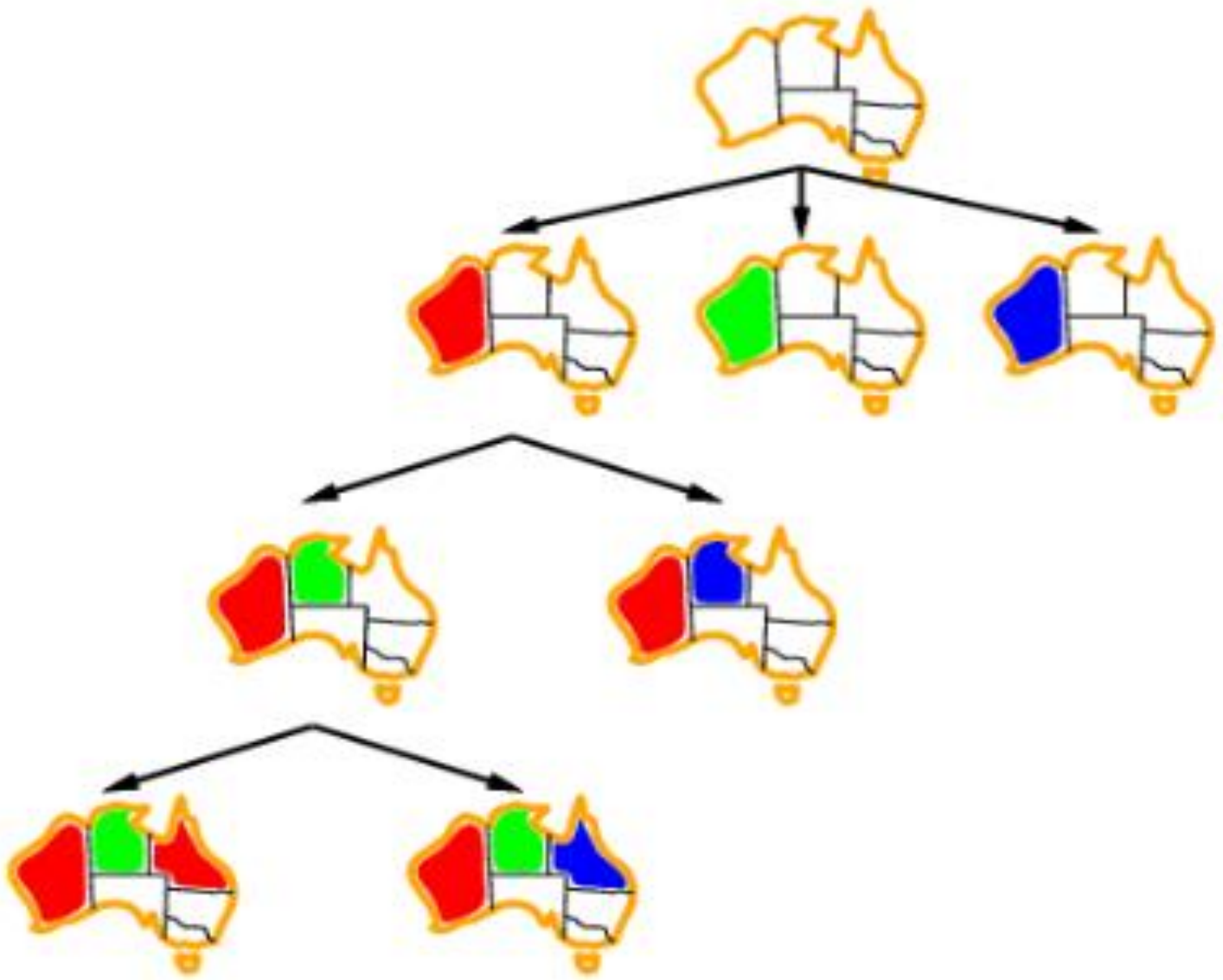
```python
N = 8 # (size of the chessboard)

def solveNQueens(board, col):
    if col == N:
        print(board)
        return True
    for i in range(N):
        if isSafe(board, i, col):
            board[i][col] = 1
            if solveNQueens(board, col + 1):
                return True
            board[i][col] = 0
    return False
```

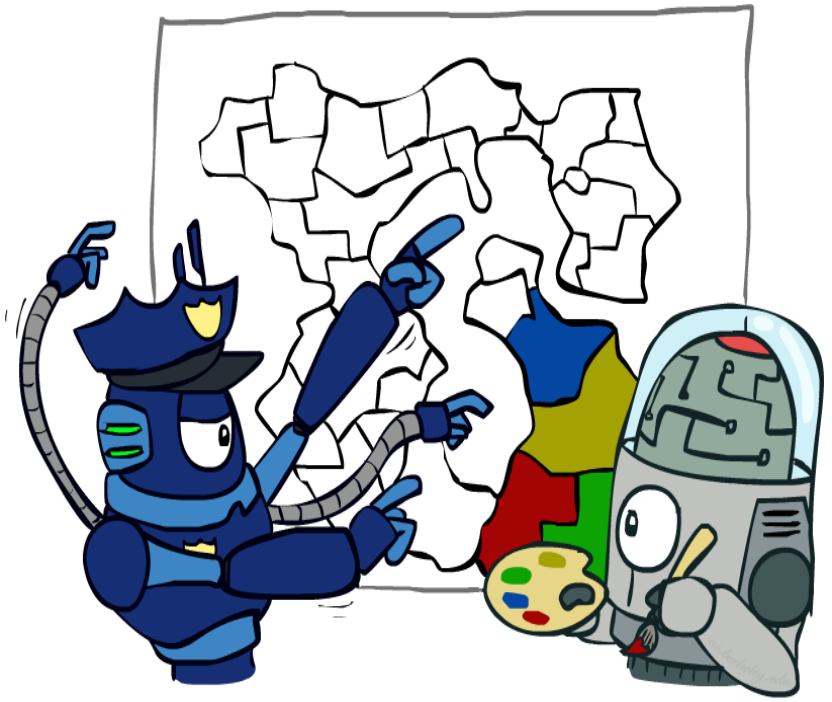
```python
def isSafe(board, row, col):
    for x in range(col):
        if board[row][x] == 1:
            return False
    for x, y in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[x][y] == 1:
            return False
    for x, y in zip(range(row, N, 1), range(col, -1, -1)):
        if board[x][y] == 1:
            return False
    return True

board = [[0 for x in range(N)] for y in range(N)]
if not solveNQueens(board, 0):
    print("No solution found")
```

Ưu-nhược điểm của Giải thuật quay lui vét cạn

- Ưu điểm: Việc quay lui là thử tất cả các tổ hợp để tìm được một lời giải. Thế mạnh của phương pháp này là nhiều cài đặt tránh được việc phải thử nhiều trường hợp chưa hoàn chỉnh, nhờ đó giảm thời gian chạy.
- Nhược điểm: Trong trường hợp xấu nhất độ phức tạp của quay lui vẫn là cấp số mũ. Vì nó mắc phải các nhược điểm sau:
  - Rơi vào tình trạng "thrashing": qúa trình tìm kiếm cứ gặp phải bế tắc với cùng một nguyên nhân.
  - Thực hiện các công việc dư thừa: Mỗi lần chúng ta quay lui, chúng ta cần phải đánh giá lại lời giải trong khi đôi lúc điều đó không cần thiết.
  - Không sớm phát hiện được các khả năng bị bế tắc trong tương lai. Quay lui chuẩn, không có cơ chế nhìn về tương lai để nhận biết đc nhánh tìm kiếm sẽ đi vào bế tắc.

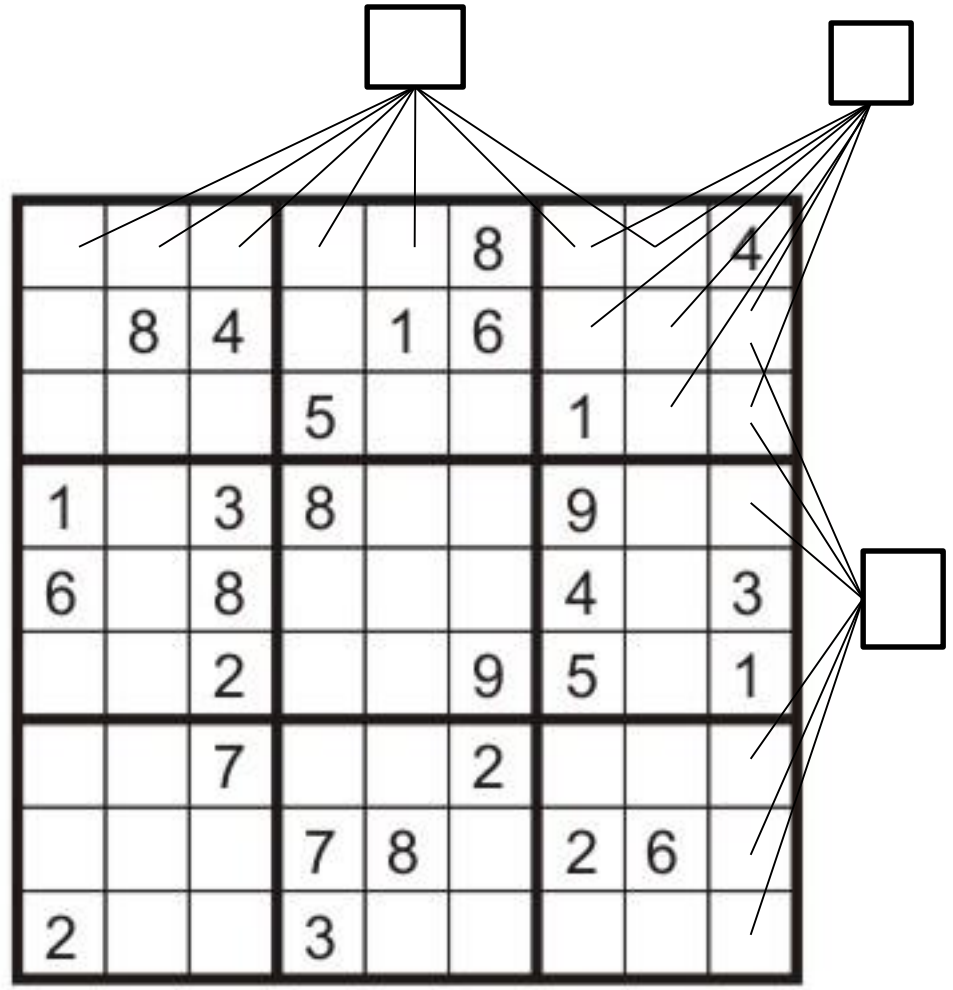
Ví dụ Giải thuật quay lui vét cạn





CSPs

Example: Sudoku



- Variables:
  - Each (open) square
- Domains:
  - {1,2,…,9}
- Constraints:

9-way alldiff for each column

9-way alldiff for each row

9-way alldiff for each region

(or can have a bunch of pairwise inequality constraints)

Cải tiến Giải thuật quay lui vét cạn

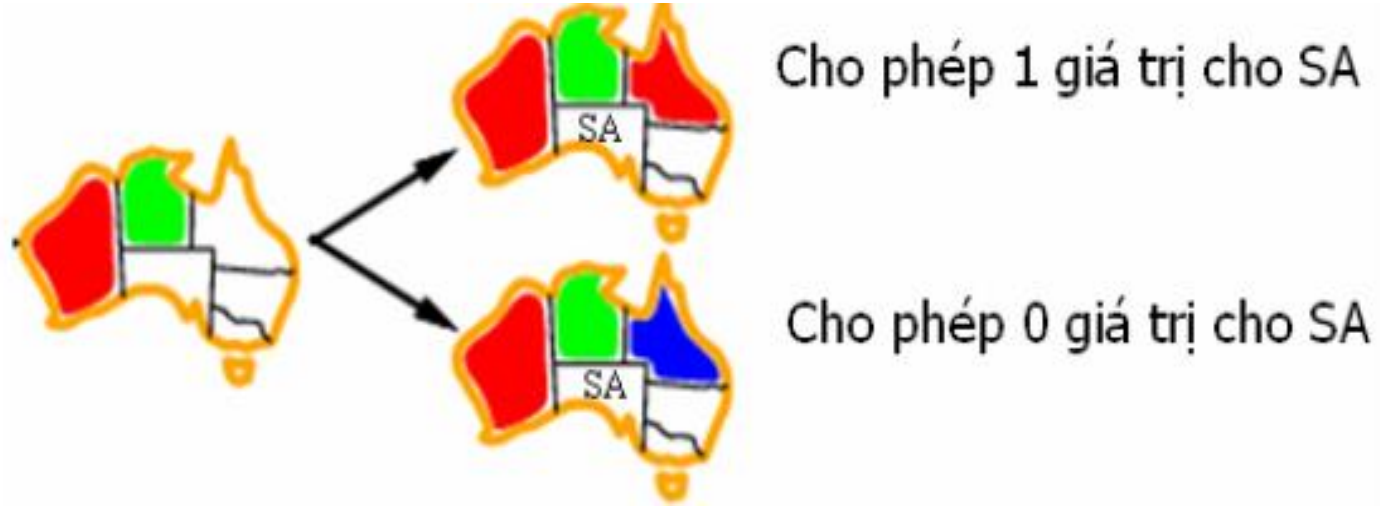
- Chọn biến tiếp theo
- Chọn thứ tự giá trị gán cho biến
- Hạn chế không gian tìm kiếm
- Lan truyền ràng buộc

Cải tiến Giải thuật quay lui vét cạn

- Chọn biến tiếp theo
  - Nguyên tắc 1: Lựa chọn biến mà miền giá trị hợp lệ còn lại là ít nhất (biến có ít lựa chọn nhất nên được chọn trước để làm giảm độ phức tạp của cây tìm kiếm)
  - Nguyên tắc 2: Lựa chọn biến tham gia vào nhiều ràng buộc nhất (gán cho biến khó thỏa mãn nhất)

Cải tiến Giải thuật quay lui vét cạn

- Chọn biến tiếp theo
- Chọn thứ tự giá trị gán cho biến:
  - Trong trường hợp bài toán yêu cầu tìm ra một lời giải và chúng ta mong muốn tìm ra lời giải trong thời gian nhanh nhất thì chúng ta sẽ lựa chọn giá trị cho biến đang xét sao cho nó ít ràng buộc đến các biến còn lại nhất.



Cải tiến Giải thuật quay lui vét cạn

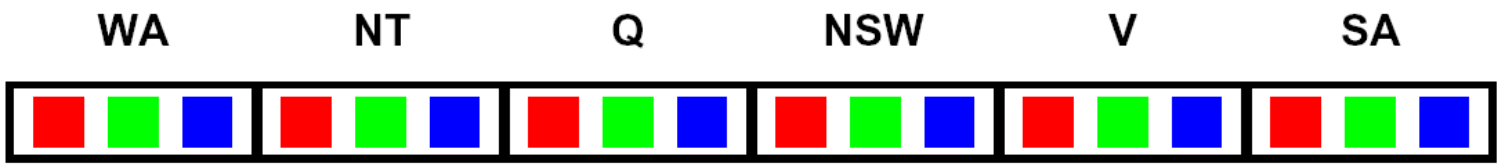
- Chọn biến tiếp theo
- Chọn thứ tự giá trị gán cho biến
- Hạn chế không gian tìm kiếm

Hạn chế không gian tìm kiếm

Keep track of domains for unassigned variables and cross off bad options

Hạn chế không gian tìm kiếm

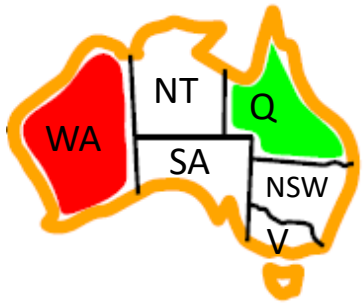
- Filtering: Theo dõi các miền cho các biến chưa được gán và gạch bỏ các tùy chọn không hợp lệ
- Forward checking: Gạch bỏ các giá trị vi phạm ràng buộc khi được thêm vào phép gán hiện có



[Demo: coloring -- forward checking]

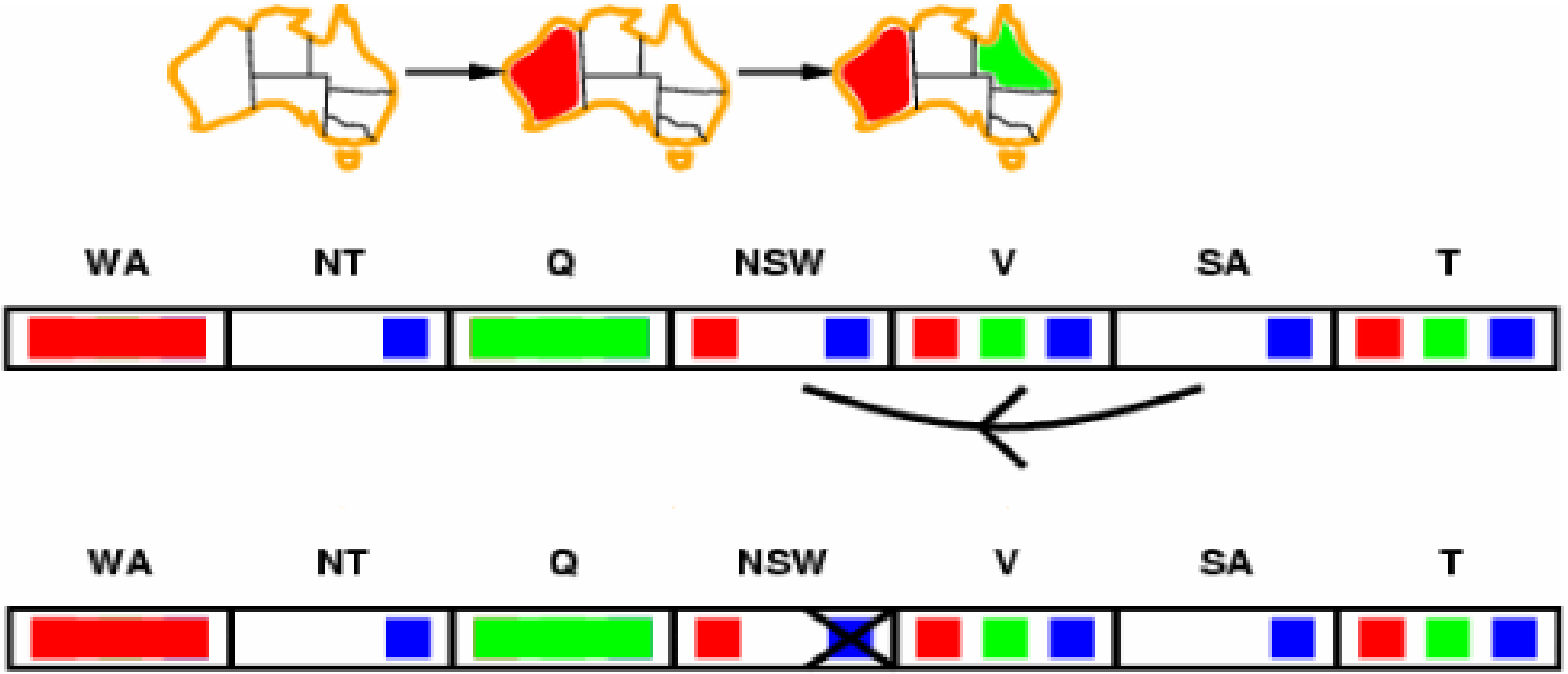
Hạn chế không gian tìm kiếm

- Forward checking truyền thông tin từ các biến được gán đến các biến chưa được gán



- Nếu ban đầu chúng ta gán WA màu đỏ
- Thì miền giá trị của các bang lân cận (NT và SA) sẽ không thể là màu đỏ được nữa.
- Nếu gán tiếp Q là màu xanh lá cây thì NT và SA chỉ còn nhận giá trị là xanh da trời
- NSW chỉ còn miền giá trị là màu đỏ

Lan truyền ràng buộc



Maintaining Arc Consistency (MAC)

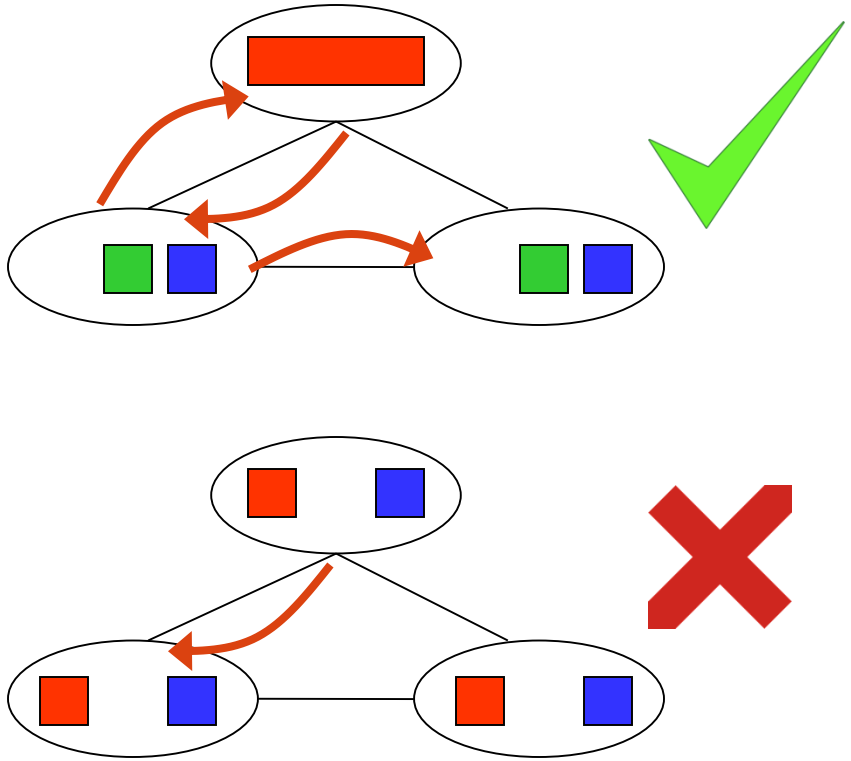
```
function AC-3( csp) returns the CSP, possibly with reduced domains
    inputs: csp, a binary CSP with variables {X_1, X_2, ..., X_n}
    local variables: queue, a queue of arcs, initially all the arcs in csp

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

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

- Runtime: $O(n^2d^3)$, can be reduced to $O(n^2d^2)$
- … but detecting all possible future problems is NP-hard – why?

Hạn chế của thuật toán Arc Consistency

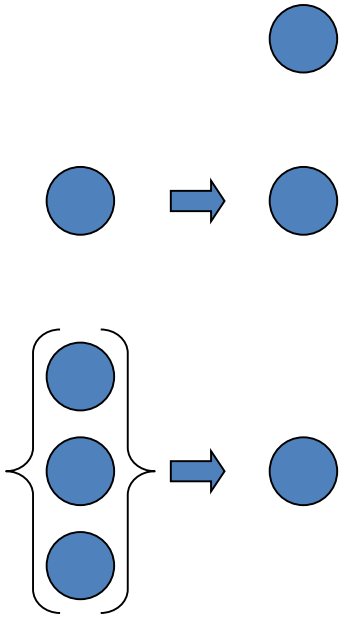
- Sau khi thực thi tính nhất quán của cung:
  - Có thể còn lại một giải pháp
  - Có thể còn lại nhiều giải pháp
  - Có thể không còn giải pháp nào (và không biết điều đó)

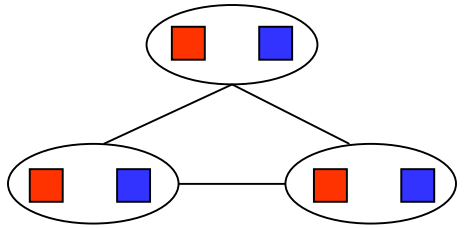


- Tính nhất quán của cung vẫn chạy bên trong tìm kiếm quay lui!

K-Consistency

- Tăng dần mức độ nhất quán
  - 1- Nhất quán (Tính nhất quán của nút): Miền của mỗi nút đơn có một giá trị đáp ứng các ràng buộc đơn nhất của nút đó
  - 2- Nhất quán (Tính nhất quán của cung): Đối với mỗi cặp nút, bất kỳ phép gán nhất quán nào cho nút này đều có thể được mở rộng sang nút kia
  - K- Nhất quán: Đối với mỗi k nút, bất kỳ phép gán nhất quán nào cho k-1 đều có thể được mở rộng sang nút thứ k.
- K cao hơn thì tốn kém hơn để tính toán
- (Bạn cần biết trường hợp k=2: tính nhất quán của cung)

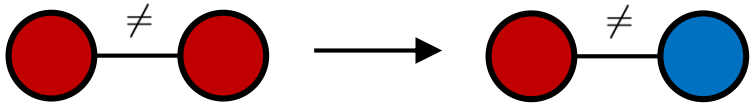




Iterative Algorithms for CSPs

- Các phương pháp tìm kiếm cục bộ thường hoạt động với các trạng thái "hoàn chỉnh", tức là tất cả các biến được gán

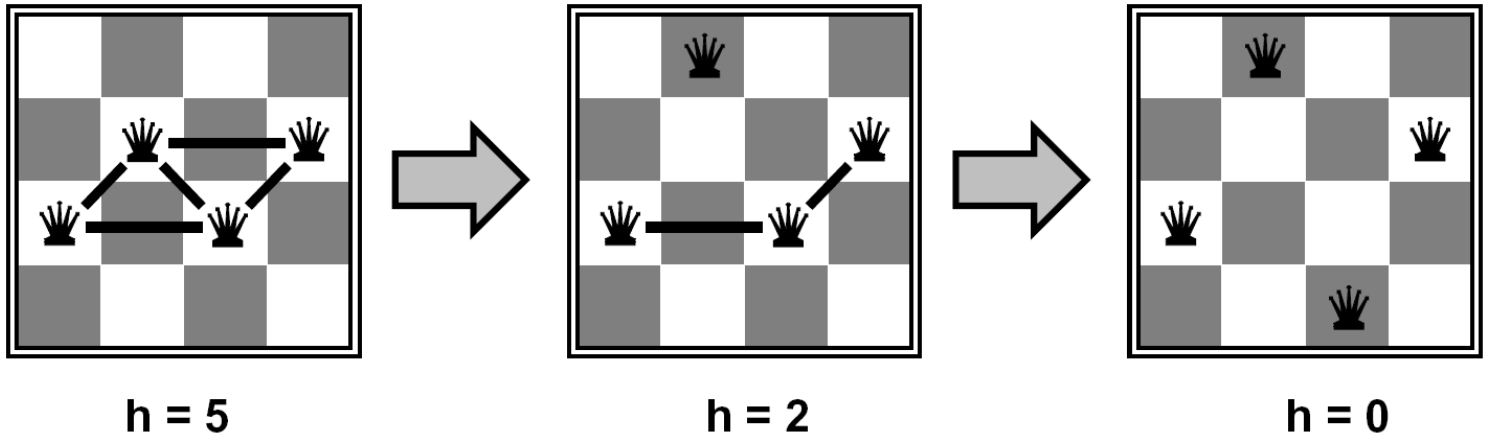
- Để áp dụng cho CSP:
  - Nhận một nhiệm vụ có ràng buộc chưa thỏa mãn
  - Các toán tử gán lại giá trị biến
  - Không có ranh giới! Sống ở rìa..



- Thuật toán: Trong khi chưa giải quyết được,
  - Lựa chọn biến: chọn ngẫu nhiên bất kỳ biến nào bị xung đột
  - Lựa chọn giá trị: min-conflicts heuristic:
  - Chọn giá trị vi phạm ít ràng buộc nhất
  - Tức là leo đồi với h(x) = tổng số ràng buộc bị vi phạm

Iterative Algorithms for CSPs

Example: 4-Queens

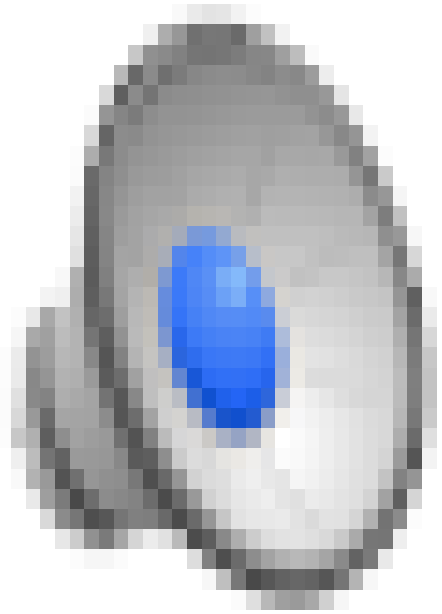


- States: 4 queens in 4 columns ($4^4 = 256$ states)
- Operators: move queen in column
- Goal test: no attacks
- Evaluation: c(n) = number of attacks

[Demo: n-queens – iterative improvement (L5D1)]
[Demo: coloring – iterative improvement]

Iterative Algorithms for CSPs

Video of Demo – n Queens



Iterative Algorithms for CSPs

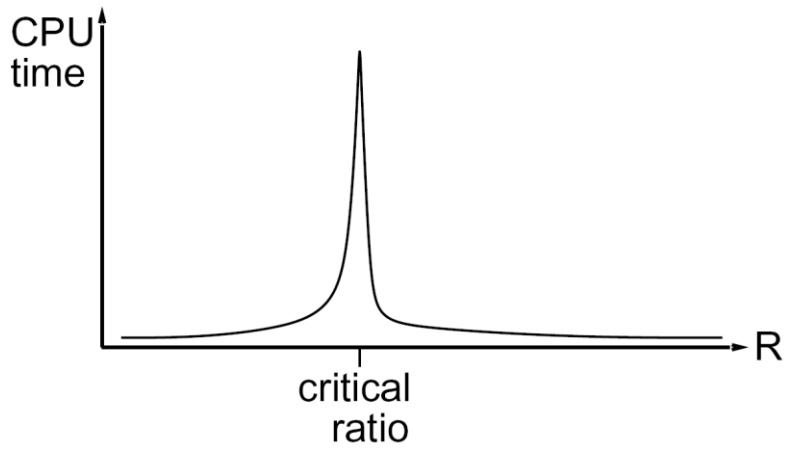
```
function MIN-CONFLICTS(csp, max_steps) returns a solution or failure
    inputs: csp, a constraint satisfaction problem
            max_steps, the number of steps allowed before giving up

    current ← an initial complete assignment for csp
    for i = 1 to max_steps do
        if current is a solution for csp then return current
        var ← a randomly chosen conflicted variable from csp.VARIABLES
        value ← the value v for var that minimizes CONFLICTS(var, v, current, csp)
        set var = value in current
    return failure
```

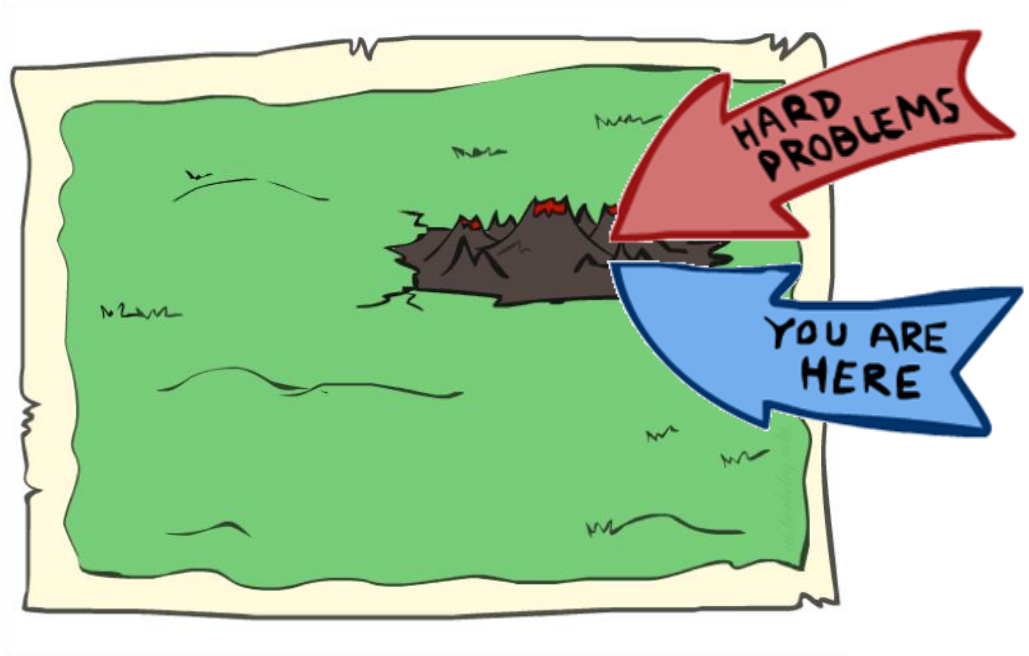
Performance of Min-Conflicts

- Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., n = 10,000,000)!

- The same appears to be true for any randomly-generated CSP *except* in a narrow range of the ratio

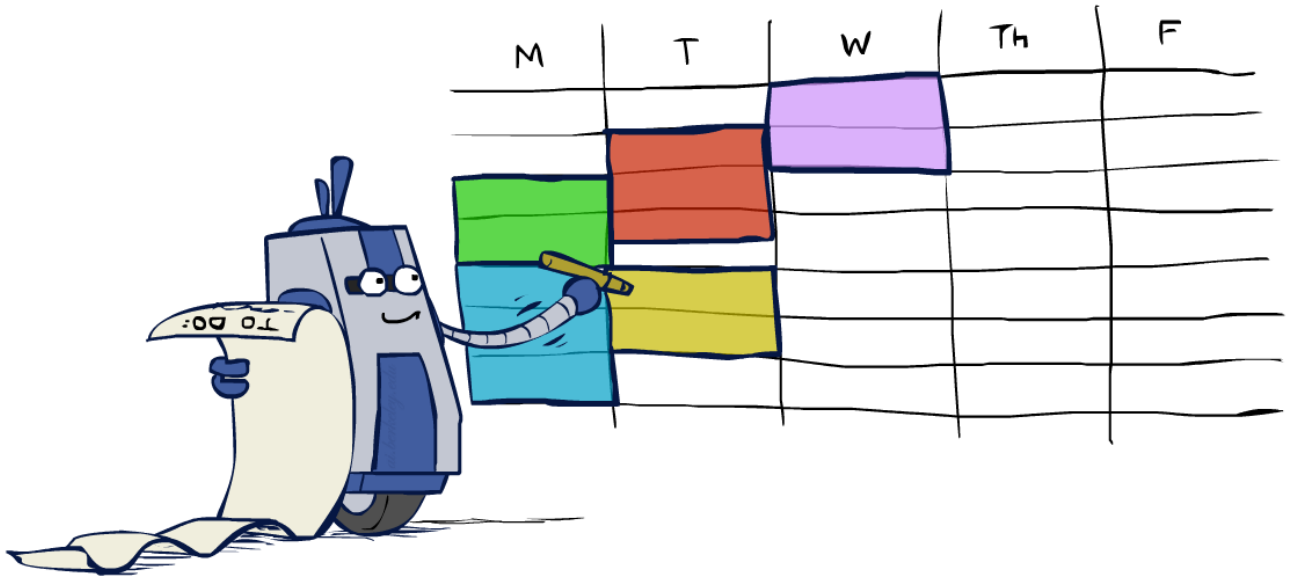
$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$





Summary: CSPs

- CSPs are a special kind of search problem:
  - States are partial assignments
  - Goal test defined by constraints
- Basic solution: backtracking search
- Speed-ups:
  - Ordering
  - Filtering
  - Structure – turns out trees are easy!



- Iterative min-conflicts is often effective in practice

Thanks for your attention!

Q&A