

Constraint Satisfaction Problems (CSPs)

Boris Velichkov

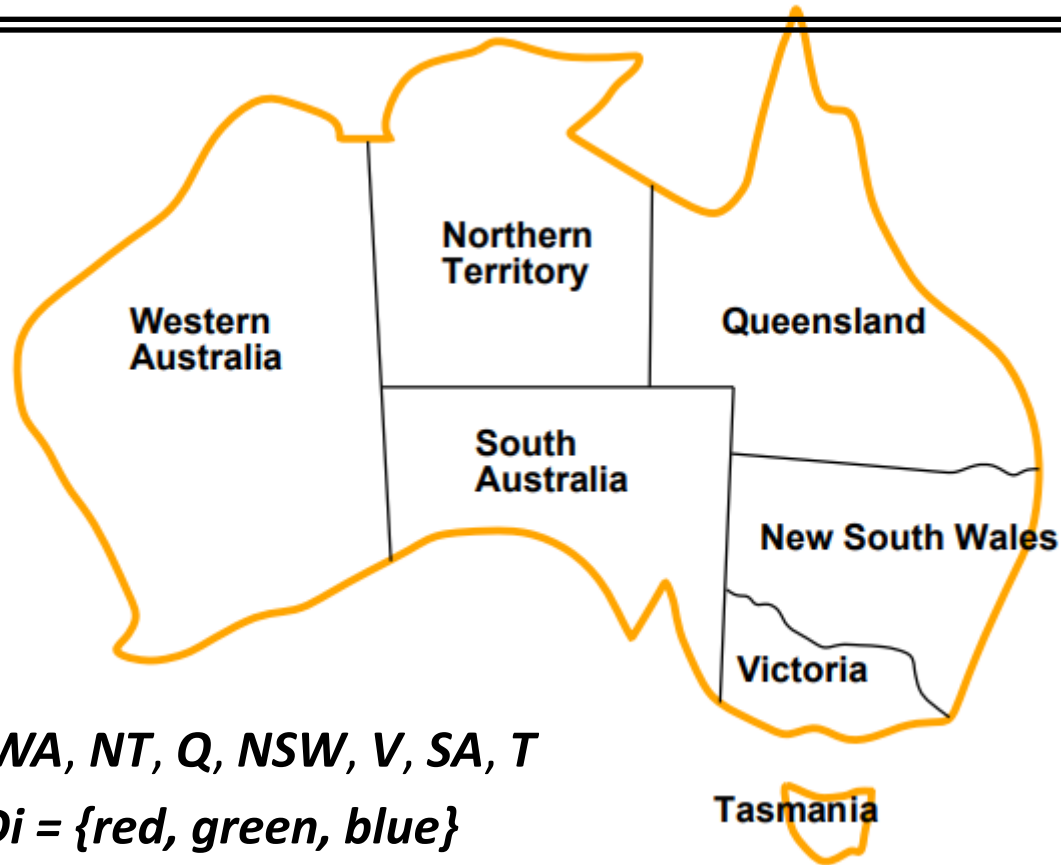
Exercises by Week

1. Introduction
2. **Uninformed (*Blind*) Search**
3. **Informed (*Heuristic*) Search**
4. **Constraint Satisfaction Problems**
5. **Genetic Algorithms**
6. **Games**
7. Introduction to Machine Learning
8. ***k*-Nearest Neighbors**
9. **Naïve Bayes Classifier**
10. **Decision Tree**
11. ***k*Means**
12. **Neural Networks**
13. *Additional Topics, Questions and Homeworks' Presentations*
14. *Additional Topics, Questions and Homeworks' Presentations*
15. *Additional Topics, Questions and Homeworks' Presentations*

Constraint Satisfaction Problems (CSPs)

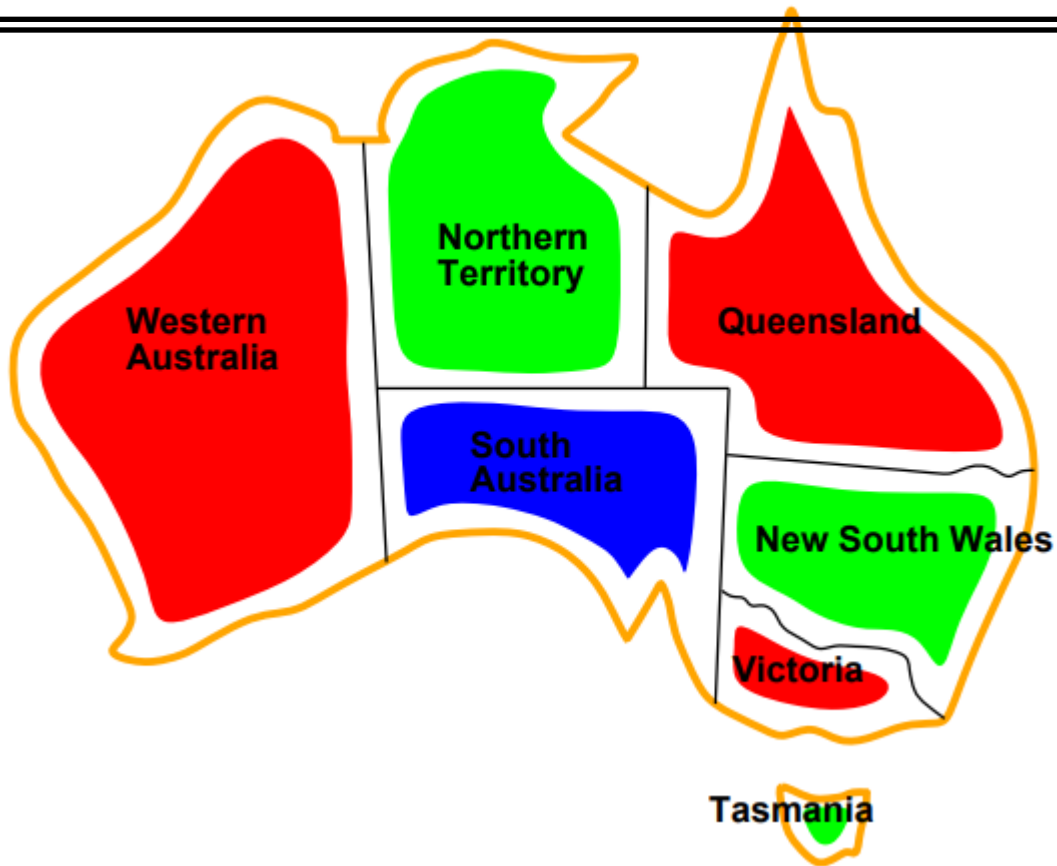
- Standard search problem:
 - state** is a “black box”—any old data structure that supports goal test, eval, successor
- CSP:
 - **state** is defined by variables X_i with values from domain D_i
 - **goal test** is a set of constraints specifying allowable combinations of *values* for subsets of *variables*
- 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: **$D_i = \{\text{red, green, blue}\}$**
- Constraints: adjacent regions must have different colors
e.g., **WA \neq NT** (if the language allows this), or
 $(WA, NT) \in \{(\text{red, green}), (\text{red, blue}), (\text{green, red}), (\text{green, blue}), \dots\}$

Example: Map-Coloring

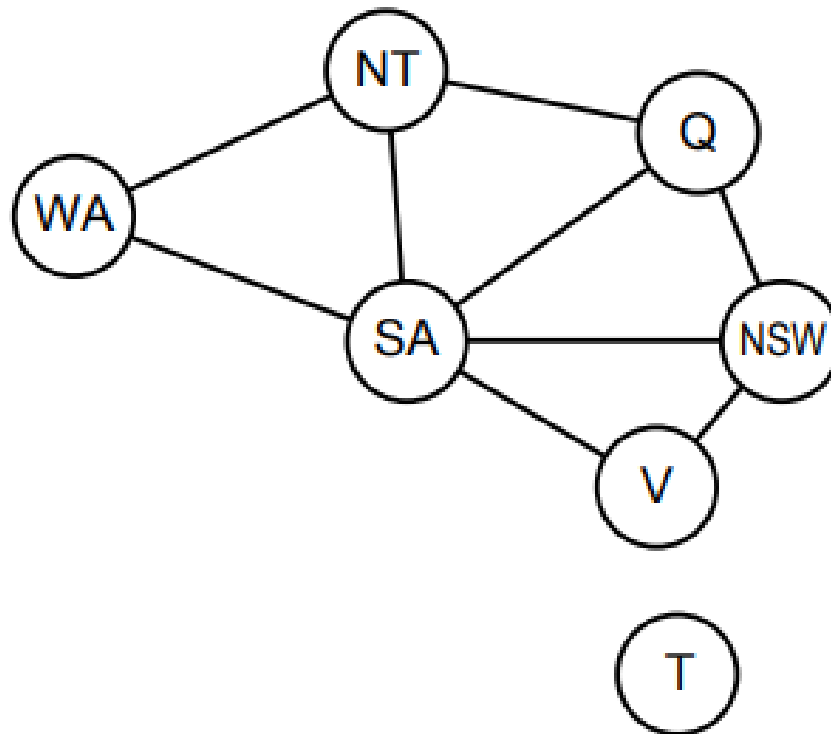


- **Solutions** are assignments satisfying all constraints, e.g.,
 $\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green\}$

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., $SA \neq \textit{green}$

Binary constraints involve pairs of variables,

e.g., $SA \neq WA$

Higher-order constraints involve 3 or more variables,

e.g., cryptarithmic column constraints

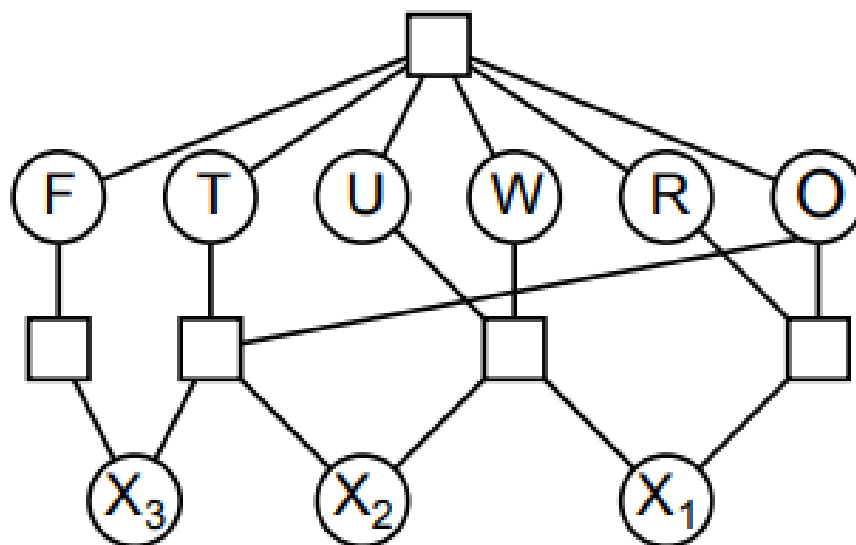
Preferences (soft constraints), e.g., \textit{red} is better than \textit{green}

often representable by a cost for each variable assignment

→ constrained optimization problems

Example: Cryptarithmic

$$\begin{array}{r}
 \text{TWO} \\
 + \text{TWO} \\
 \hline
 \text{FOUR}
 \end{array}$$



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

$\text{alldiff}(F, T, U, W, R, O)$

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

928

928

1856

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

CSPs: The most used techniques

The most used techniques are variants of:

- **Backtracking** (*Depth-First Search*)
- **Constraint Propagation**
 - *Arc Consistency*
- **Local Search**
 - *minConflict (Hill Climbing)*

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.
⇒ 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
⇒ use depth-first search
 - 3) Path is irrelevant, so can also use complete-state formulation
 - 4) $b = (n - \ell)d$ at depth ℓ , hence $n!d^n$ leaves!!!! 😞

Backtracking search

Variable assignments are *commutative*, i.e.,

$[WA = \text{red} \text{ then } NT = \text{green}]$ same as $[NT = \text{green} \text{ then } WA = \text{red}]$

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

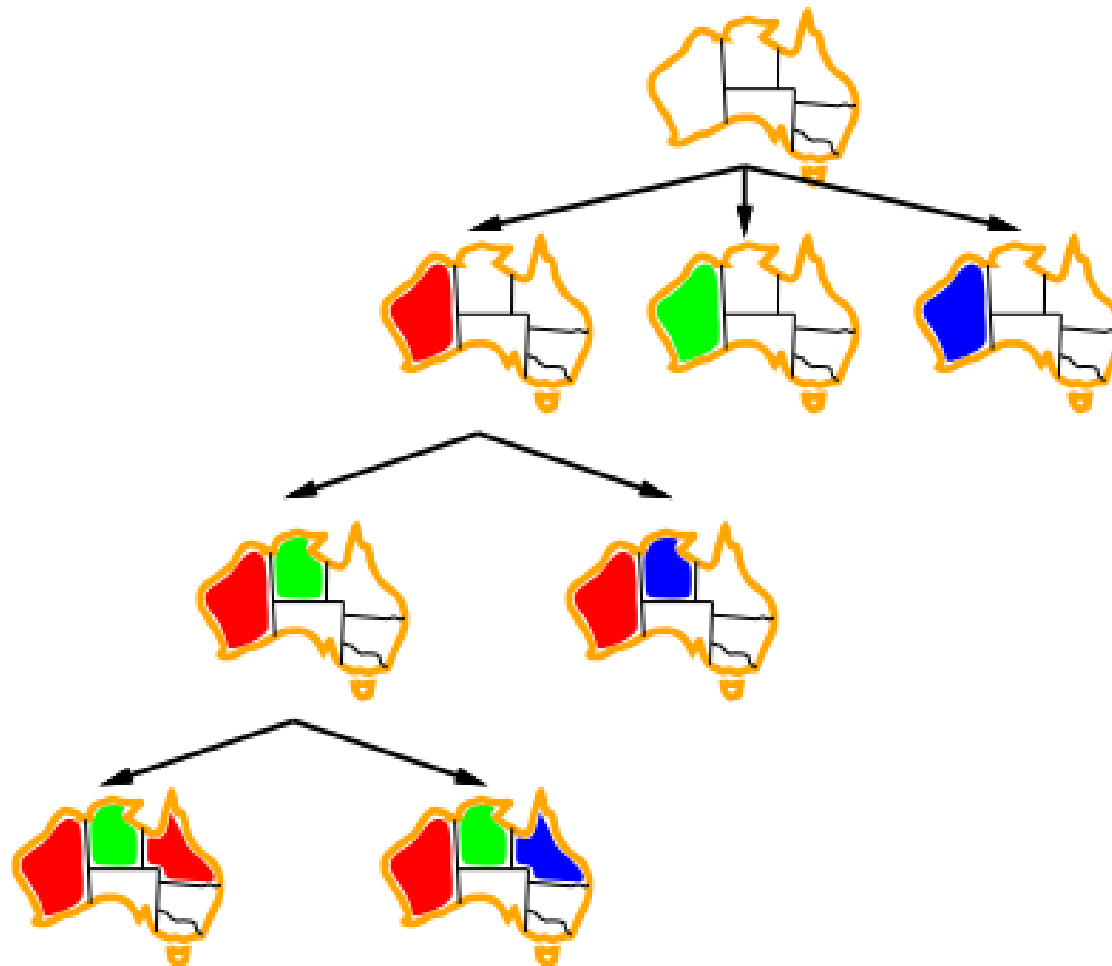
$\Rightarrow b = d$ and there are d^n leaves

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 example



Improving backtracking efficiency

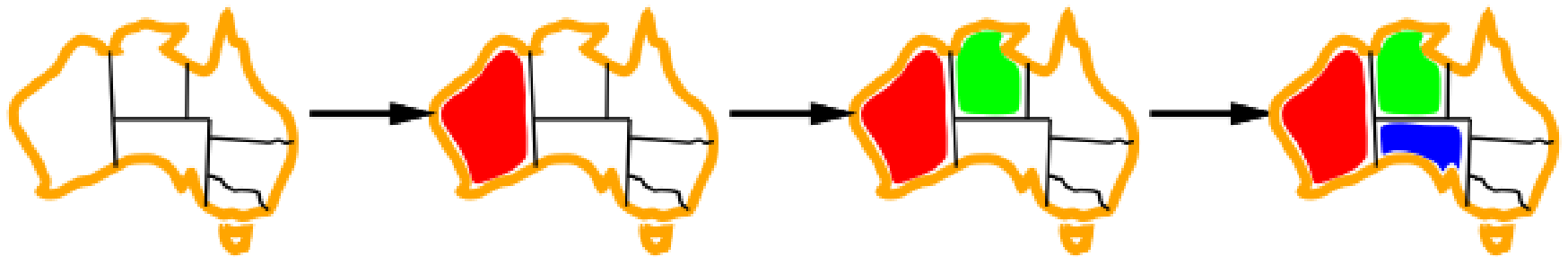
General-purpose methods can give huge gains in speed:

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

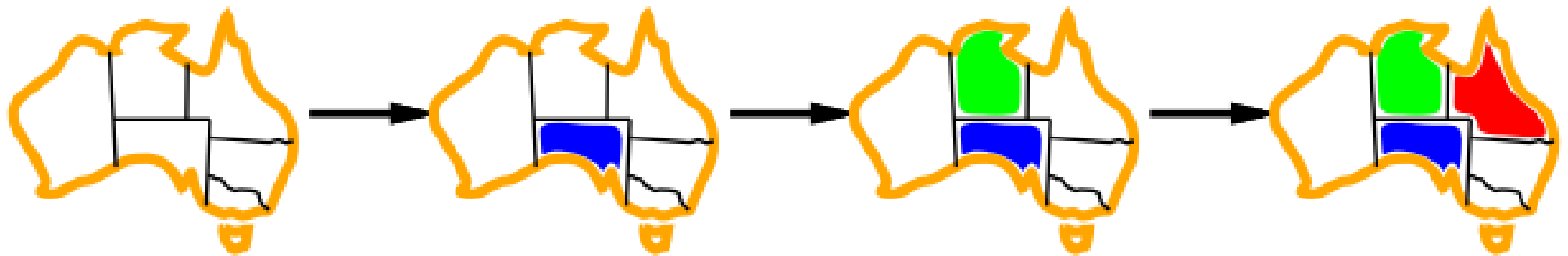


Degree heuristic

Tie-breaker among MRV variables

Degree heuristic:

choose the variable with the most constraints on remaining variables

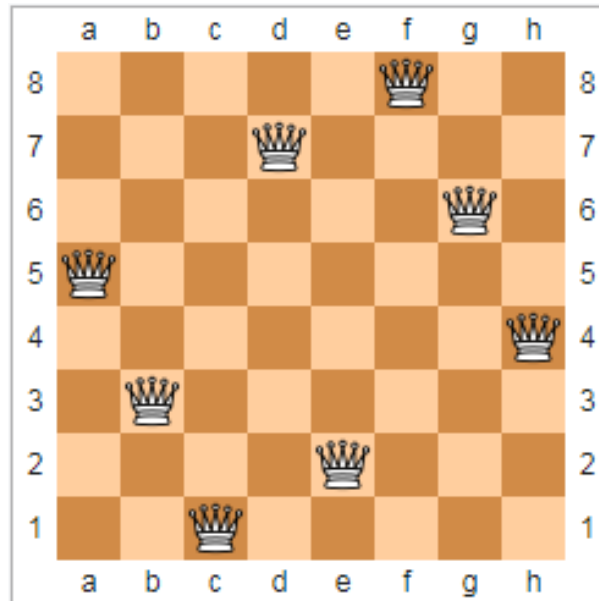


More In Lectures

- Forward checking
- Arc Consistency

8 Queens

- The ***eight queens puzzle*** is the problem of placing eight *chess queens* on an **8×8** chessboard so that no two queens threaten each other; thus, a solution requires that no two queens share the same row, column, or diagonal. The eight queens puzzle is an example of the more general **n queens problem** of placing **n** non-attacking queens on an **$n \times n$** chessboard, for which solutions exist for all natural numbers **n** with the exception of **$n = 2$** and **$n = 3$** .



Homework: N Queens

Place **N** chess queens on an **NxN** chessboard so that no two queens threaten each other. In other words the solution requires that no two queens share the same row, column, or diagonal. Use the **MinConflicts** algorithm to solve the problem.

Input: Integer N - the number of queens to be located.

** Requirement to work for N = 10,000*

Output: Print in the console the board by designating a queen with * and an empty cell with _

Sample input:

4

Sample output:

```
* _ _ _
_ _ _ *
_ _ _ _
* _ _ _
_ _ * _
_ _ _ _
```

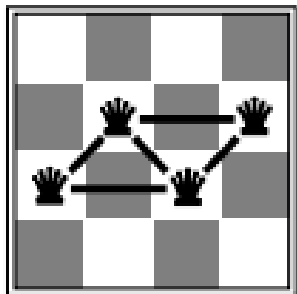
Example: 4-Queens

States: 4 queens in 4 columns ($4^4 = 256$ states)

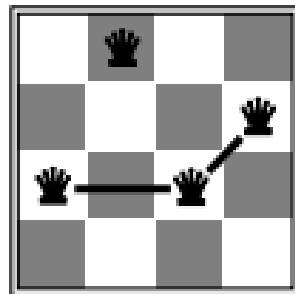
Operators: move queen in column

Goal test: no attacks

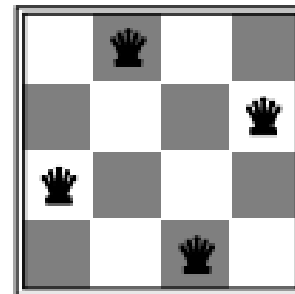
Evaluation: $h(n)$ = number of attacks



$h = 5$

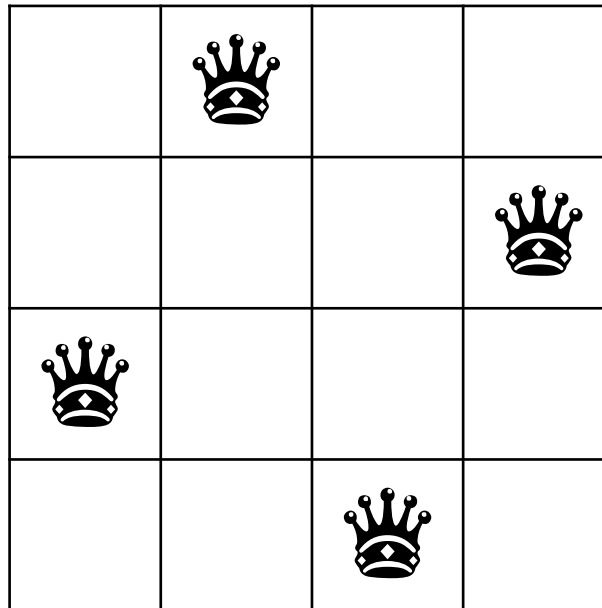


$h = 2$

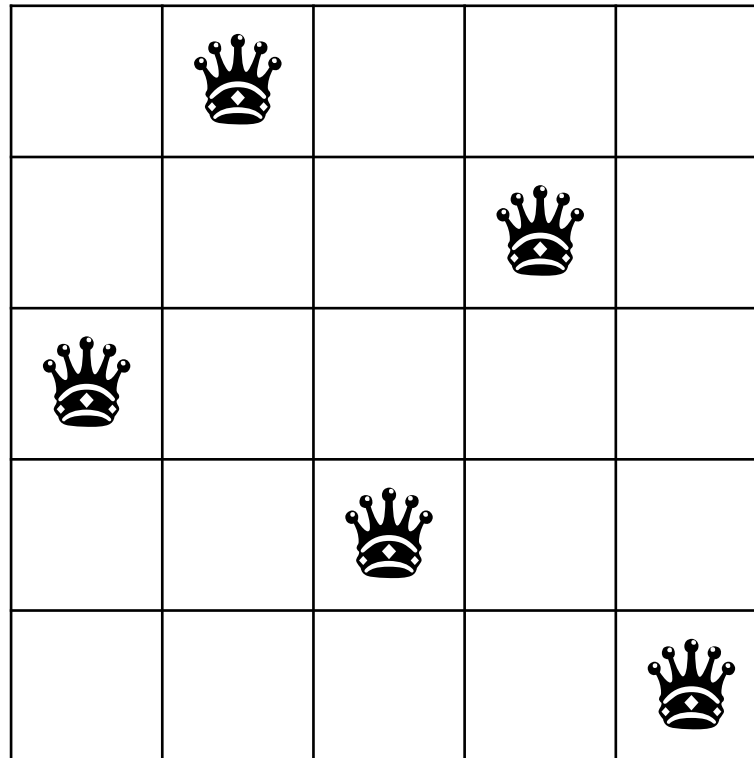


$h = 0$

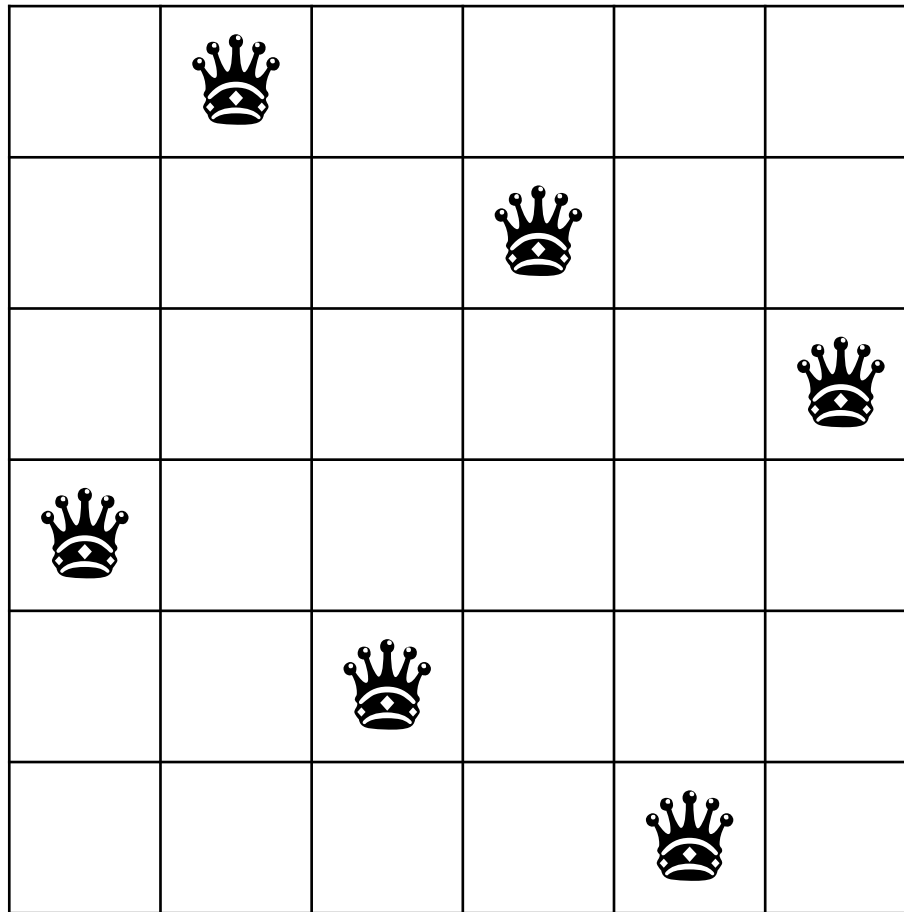
N Queens: $N=4$







N Queens: $N=5$



N Queens: $N=6$



N Queens: $N=4$

<i>value</i>	[2, 0, 3, 1]			
<i>index</i>	0	1	2	3
0				
1				
2				
3				

N Queens: $N=4$

$y \backslash x$	0	1	2	3			\backslash d1: $y1 - x1 = y2 - x2$ / d2: $y1 + x1 = y2 + x2$
0	(0,0)	(1,0)	(2,0)	(3,0)	$3 - 0 = 3$	6	
1	(0,1)	(1,1)	(2,1)	(3,1)	$2 - 0 = 3 - 1 = 2$	5	
2	(0,2)	(1,2)	(2,2)	(3,2)	$1 - 0 = 2 - 1 = 3 - 2 = 1$	4	
3	(0,3)	(1,3)	(2,3)	(3,3)			
				$0 - 0 =$ $0 - 1 =$ $0 - 2 =$ $0 - 3 =$			$2n - 1 = 2*4 - 1 = 7$ Diagonals We present each of the obtained values in an array of 7 elements (indices from 0 to 6).
		$1 - 2 =$ $1 - 3 =$	$1 - 1 =$ $1 - 2 =$ $2 - 3 =$	$1 - 1 =$ $2 - 2 =$ $3 - 3 =$			
	0	1	2	3			

N Queens: Pseudocode

```
solve(N) {  
    // Putting the queen on the row with min conflicts  
    queens[] = init(N)  
    iter = 0  
    while(iter++ <= k*N) {  
        // Randomly if two or more!  
        col = getColWithQueenWithMaxConf()  
        // Randomly if two or more!  
        row = getRowWithMinConflict(col)  
        // index=col & value=row  
        queens[col] = row  
    }  
    if(hasConflicts()) {  
        // Restart  
        solve(N)  
    }  
}
```