

# *Constraint Satisfaction Problems - CSP*

March 23, 2017



## n-queens

- Problem: place  $n$  queens in a board  $n \times n$  so that no queens attack each other in the rows, columns or diagonals
- 2 ways of solving the problem: **incremental** and **complete**
- Possible states (a) and moves (b):
  1. Incremental 1
    - a) any arrangement of 0 to  $n$  queens in the board
    - b) add 1 queen to any position in the board
  2. Incremental 2
    - a) arrangements of 0 to  $n$  queens with no attacks
    - b) add 1 queen to the next leftmost empty column such that this does not attack the others already placed
  3. Completo
    - a) arrangements of  $n$  queens, 1 em cada coluna
    - b) mover qq rainha atacada para outra posição na mesma coluna

# Cryptarithmic

|       |          |            |
|-------|----------|------------|
|       | SOLUTION |            |
| FORTY | 29786    | com: F = 2 |
| + TEN | 850      | 0 = 9      |
| + TEN | 850      | R = 7 etc  |
| ----- | -----    |            |
| SIXTY | 31486    |            |

# Cryptarithmic

- states: set of letters that are replaced by digits
- operators (moves): replace all occurrences of a letter by a digit that was never used before
- possible rules to select digits: numerical order to avoid repetitions, more strict obeying the mathematical properties of the problem
- final state: solution contains only digits and represent a correct sum
- cost of the solution: zero









## Other Approach and Representation

- Constraint Satisfaction
  - ▶ special type of problem that satisfies structural properties besides the basic requirements for general search problems
  - ▶ states: set of variables
  - ▶ variables can take values from a domain: set of possible values a variable can take (discrete/continuous, finite/infinite)
  - ▶ initial state: all variables with possible initial values from the domain
  - ▶ final state: final assignment of variable-value that does not violate the problem constraints
  - ▶ maximum search tree depth: number of variables
- Explicit representation for constraints (Portuguese translation: restrições)

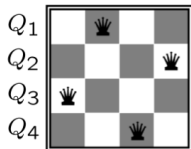




## Example CSP: N-queens

- Formulation 2:
  - ▶ Variables:  $Q_k$
  - ▶ Domains:  $\{1,2,3,\dots\}$
  - ▶ Constraints:

$\forall i, j$  non-threatening( $Q_i, Q_j$ ) or  
 $(Q_1, Q_2) \in \{(1, 3), (1, 4), \dots\}$   
 ...





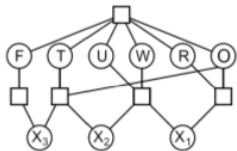
## Example CSP: Cryptarithmic

- Variables (circles): F T U W R O X1 X2 X3

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

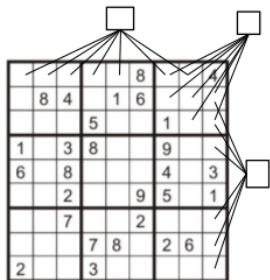
- Domains:  $\{0,1,2,3,4,5,6,7,8,9\}$
- Constraints (boxes):

`alldiff(F,T,U,W,R,O)`  
`0 + 0 = R + 10 . X1`  
`...`



## Example CSP: Sudoku

- Variables: Each (open) square
- Domains:  $\{1, 2, \dots, 9\}$
- Constraints:
  - ▶ 9-way alldiff for each column
  - ▶ 9-way alldiff for each row
  - ▶ 9-way alldiff for each region







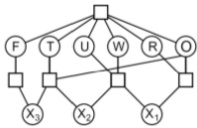


## Standard Search Formulation

- Standard search formulation of CSPs (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
  - ▶ Goal test: the current assignment is complete and satisfies all constraints

# Search Methods

- What does BFS do?
- What does DFS do?



# Constraint Satisfaction: improving search

- n-queens
  - ▶ variables: possible positions in the board
  - ▶ constraints: no queen can attack each other
  - ▶ initial variable values:  $V_1$  in  $1..n$ ,  $V_2$  in  $1..n$  etc, with  $n$  the width of the board

# Constraint Satisfaction: possible algorithms

- aloca uma nova rainha para uma nova coluna a cada nível (solução incremental)
- complexidade:  $n^n \approx \prod D_i = D_1 \times D_2 \times \dots \times D_n$
- fator de ramificação:  $n$
- fator máximo de ramificação:  
 $n \times n = \sum D_i = D_1 + D_2 + \dots + D_n$ 
  - ▶ se todas as variáveis fossem instanciadas com todos os valores possíveis no primeiro nível da árvore

# Constraint Satisfaction: Possible algorithms

n = 8

(0,0,0,0,0,0,0,0)

|   |                   |                   |                   |      |
|---|-------------------|-------------------|-------------------|------|
| N | (1,0,0,0,0,0,0,0) | (0,1,0,0,0,0,0,0) | (0,0,1,0,0,0,0,0) | .... |
| i | (2,0,0,0,0,0,0,0) | (0,2,0,0,0,0,0,0) | (0,0,2,0,0,0,0,0) | .... |
| v | .....             |                   |                   |      |
| 1 | (8,0,0,0,0,0,0,0) | (0,8,0,0,0,0,0,0) | (0,0,8,0,0,0,0,0) | .... |
| N | (1,1,0,0,0,0,0,0) | (0,1,1,0,0,0,0,0) | (1,0,1,0,0,0,0,0) | .... |
| i | (2,2,0,0,0,0,0,0) | (0,2,2,0,0,0,0,0) | (2,0,2,0,0,0,0,0) | .... |
| v | .....             |                   |                   |      |
| 2 | (8,8,0,0,0,0,0,0) | (0,8,8,0,0,0,0,0) | (8,0,8,0,0,0,0,0) | .... |



## Constraint Satisfaction: Possible algorithms

- solution: test the constraint whenever a new queen is placed in the board
- it is not the best solution either
- Suppose that we managed to place 6 queens that are safe. This placement may threat the 8th queen and the algorithm does not know!
- DFS will end up testing ALL possibilities for placing the 7th queen in the board!
- Class of algorithms: “constrain-and-generate”



## Constraint Satisfaction: Possible algorithms

- solution: Look ahead!
- *Forward Checking*: checks if the other unassigned variables domains are consistent with the new partial solution, removing from the domain of other variables all values that violate the constraints
- General Lookahead (arc-consistency): besides executing forward checking, checks if the new set of domains conflict with each other







# Solutions to n-queens WITHOUT searching

(Explicit Solutions to the N-Queens Problem for all N, by Bo Bernhardsson)

- for n even and not of the form  $6k + 2$ 
  - ▶  $(j, 2j)$
  - ▶  $\frac{n}{2} + j, 2j - 1$ , for  $j = 1, 2, \dots, \frac{n}{2}$
- for n even, but not in the form  $6k$ 
  - ▶  $(j, 1 + [2(j - 1) + \frac{n}{2} - 1 \pmod{n}])$
  - ▶  $(n + 1 - j, n - [2(j - 1) + \frac{n}{2} - 1 \pmod{n}])$ , for  $j = 1, 2, \dots, \frac{n}{2}$
- for n odd
  - ▶ Use any case above for n-1 and extend with 1 queen  $(n, n)$

# Solutions for n-queens **WITHOUT** searching

- Other references:
  - ▶ The n-Queens Problem, by Igor Rivin, Ilan Vardi, and Paul Zimmermann
  - ▶ A simplified solution of the N queens' problem, by Matthias Reichling
- Observation: find **all** solutions for the n-queens problem is not trivial :(

# Constraint Satisfaction: other algorithms

- for infinite domains: linear programming, simplex, revised simplex, convex hull, Gauss elimination
- for finite domains: forward checking, lookahead, arc-consistency in general
- for finite domains, two problems to solve:
  - ▶ choice of the *variable*:
    - *most-constrained*: smaller domain
    - *most constraining*: constrains the domain of other variables as much as possible
  - ▶ choice of the *value* for a variable:
    - *first-fail* principle.
    - *least constraining*: value that leaves more choices open to other variables

# Constraint Satisfaction

- most-constrained: allows to solve n-queens with n equals to 100.
- pure forward checking: solves at most 30
- least-constraining value: allows to solve n-queens with n equals to 1000.



# Algorithms

- Backtracking (systematic search)
- Constraint propagation: k-consistency
- All that use heuristics to order variables and their values
- Backjumping and dependency-directed backtracking

# Basic Algorithm

CSP-BACKTRACKING(PartialAssignment a)

  If a is complete then return a

  X ← select an unassigned variable

  D ← select an ordering for the domain of X

  For each value v in D do

    If v is consistent with a then

      Add (X = v) to a

      result ← CSP-BACKTRACKING(a)

      If result ≠ failure then return result

      Remove (X = v) from a

  Return failure

Initial call: CSP-BACKTRACKING({})

Solves n-queens till  $n \approx 25$

Combination of constraint propagation and heuristics can solve instances of size 1000

Random algorithms with min-conflicts plus parallelization can solve 3M queens in less than a minute! (see recommended reading in the discipline page)