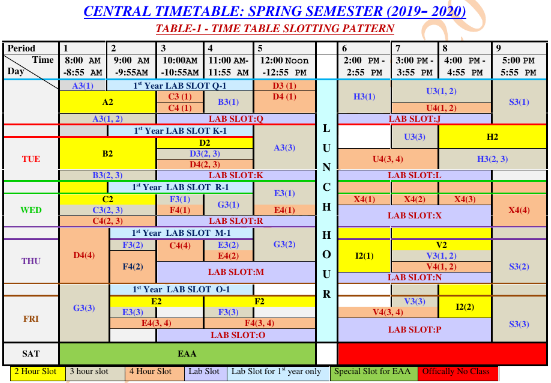
Artificial Intelligence: Foundations & Applications

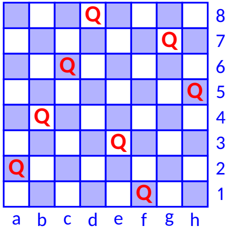
Introduction to Constraint Satisfaction Problem

Prof. Partha P. Chakrabarti & Arijit Mondal
Indian Institute of Technology Kharagpur

Examples of CSP

- Crossword puzzle
- N-queens on chess board
- Knapsack
- Assembly scheduling
- Operations research
- Map coloring
- Time tabling
- Airline/train scheduling
- Cryptic puzzle
- Boolean satisfiability
- Car sequencing
- Scene labeling
- etc.





CSP formulation

- **Variables**
  - **A set of *decision* variables** $x_1, x_2, \ldots, x_n$

CSP formulation

- **Variables**
  - **A set of *decision* variables** $x_1, x_2, \ldots, x_n$
- **Domain of variables**
  - **Each variable has a domain (discrete or continuous)** $D_1, D_2, \ldots, D_n$ **from which it can take a value.**

CSP formulation

- **Variables**
  - **A set of *decision* variables** $x_1, x_2, \ldots, x_n$
- **Domain of variables**
  - **Each variable has a domain (discrete or continuous)** $D_1, D_2, \ldots, D_n$ **from which it can take a value.**
- **Satisfaction constraint**
  - **A finite set of satisfaction constraints** $C_1, C_2, \ldots, C_m$
  - **A constraint can be unary, binary or among many variables. Given a value of variables, any constraint will yield *yes* or *no* only**

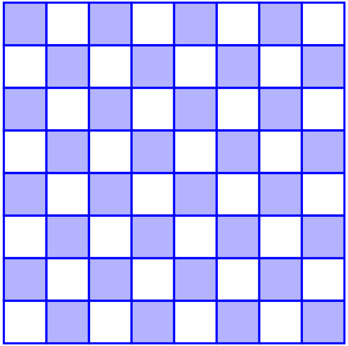
CSP formulation

- **Variables**
  - **A set of *decision* variables** $x_1, x_2, \ldots, x_n$
- **Domain of variables**
  - **Each variable has a domain (discrete or continuous)** $D_1, D_2, \ldots, D_n$ **from which it can take a value.**
- **Satisfaction constraint**
  - **A finite set of satisfaction constraints** $C_1, C_2, \ldots, C_m$
  - **A constraint can be unary, binary or among many variables. Given a value of variables, any constraint will yield *yes* or *no* only**
- **Cost function for optimization (optional)**
  - **A set of optimization functions (typically** $\min, \max$**)** $O_1, O_2, \ldots, O_p$

CSP formulation

- Variables
  - A set of *decision* variables $x_1, x_2, \ldots, x_n$
- Domain of variables
  - Each variable has a domain (discrete or continuous) $D_1, D_2, \ldots, D_n$ from which it can take a value.
- Satisfaction constraint
  - A finite set of satisfaction constraints $C_1, C_2, \ldots, C_m$
  - A constraint can be unary, binary or among many variables. Given a value of variables, any constraint will yield *yes* or *no* only
- Cost function for optimization (optional)
  - A set of optimization functions (typically $\min, \max$) $O_1, O_2, \ldots, O_p$
- Solution
  - A consistent assignment of domain values to each variable so that all constraints are satisfied and the optimization criteria (if any) are met.

N-Queens

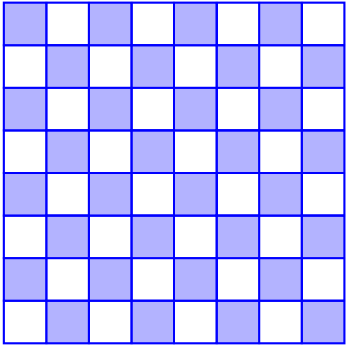


Need to place N-queens on this board

Rules:

- No queens are attacking each other

N-Queens



Need to place N-queens on this board

Rules:

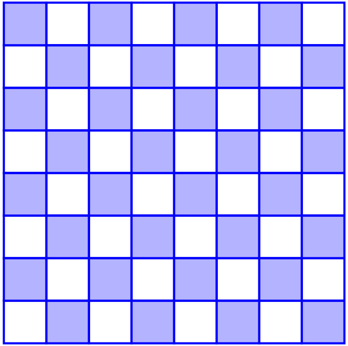
- No queens are attacking each other

- Variables: x_{ij} - queen is in cell (i,j),
- Domains: $D_{ij} \in \{0,1\}$
- Constraints: $\sum_i x_{ij} = 1, \sum_j x_{ij} = 1, \sum_{i,j} x_{ij} = N,$

  $x_{ij} + x_{(i+k)(j+k)} \leq 1, \quad x_{ij} + x_{(i+k)(j-k)} \leq 1,$

  k is in appropriate range
- Search space $2^{64} = 18,446,744,073,709,551,616$

N-Queens (alternative model)

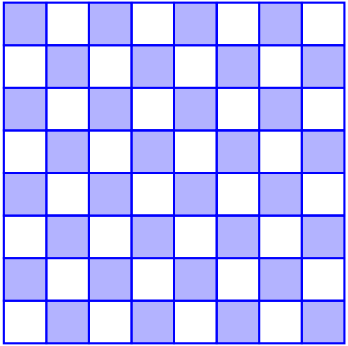


Need to place N-queens on this board

Rules:

- No queens are attacking each other

N-Queens (alternative model)



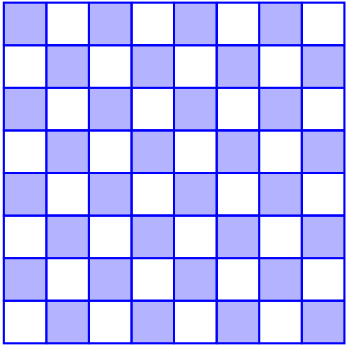
Need to place N-queens on this board

Rules:

- No queens are attacking each other

- Variables: x_i
- Domains: $D_i \in \{1, 2, \ldots, 8\}$
- Constraints: $\ldots$
- Search space $8^8 = 16,777,216$

N-Queens (alternative model)



Need to place N-queens on this board

Rules:
- No queens are attacking each other

- Variables: x_i
- Domains: $D_i \in \{1, 2, \ldots, 8\}$
- Constraints: $\ldots$
- Search space $8^8 = 16,777,216$

Other variants:
- At least a queen on the main diagonal
- Two queens on the two main diagonals
- Enumeration of all solutions

Examination schedule

Student	Subjects
S_1	C_1, C_2, C_3
S_2	C_2, C_3, C_4
S_3	C_3, C_4
S_4	C_3, C_4, C_5
S_5	C_1, C_5, C_6

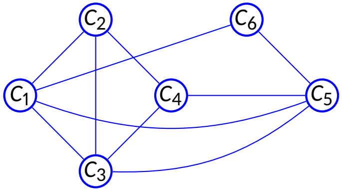
Examination schedule

Student	Subjects
S_1	C_1, C_2, C_3
S_2	C_2, C_3, C_4
S_3	C_3, C_4
S_4	C_3, C_4, C_5
S_5	C_1, C_5, C_6

Is it possible to conduct all these exams in 3 days assuming one exam per day?

Examination schedule

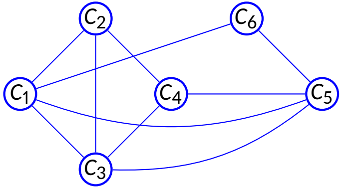
Student	Subjects
S_1	C_1, C_2, C_3
S_2	C_2, C_3, C_4
S_3	C_3, C_4
S_4	C_3, C_4, C_5
S_5	C_1, C_5, C_6



Is it possible to conduct all these exams in 3 days assuming one exam per day?

Examination schedule

Student	Subjects
S_1	C_1, C_2, C_3
S_2	C_2, C_3, C_4
S_3	C_3, C_4
S_4	C_3, C_4, C_5
S_5	C_1, C_5, C_6

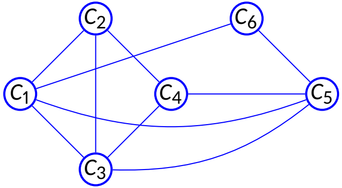


Is it possible to conduct all these exams in 3 days assuming one exam per day?

- Variables: x_i - slot for subject C_i
- Domains: $D_i \in \{1, 2, 3\}$
- Constraints: $x_1 \neq x_2, x_1 \neq x_3, \ldots$

Examination schedule

Student	Subjects
S_1	C_1, C_2, C_3
S_2	C_2, C_3, C_4
S_3	C_3, C_4
S_4	C_3, C_4, C_5
S_5	C_1, C_5, C_6



Is it possible to conduct all these exams in 3 days assuming one exam per day?

- Variables: x_i - slot for subject C_i
- Domains: $D_i \in \{1, 2, 3\}$
- Constraints: $x_1 \neq x_2, x_1 \neq x_3, \ldots$

Graph coloring problem.

Airport gate scheduling

Flight	Arrv. time	Dept. time
F1	0715	0815
F2	0800	0900
F3	0830	0930
F4	0845	0945
F5	0915	1015
F6	0845	0945

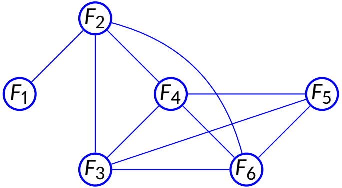
Airport gate scheduling

Flight	Arrv. time	Dept. time
F1	0715	0815
F2	0800	0900
F3	0830	0930
F4	0845	0945
F5	0915	1015
F6	0845	0945

Is it possible to schedule all flights using 3 gates?

Airport gate scheduling

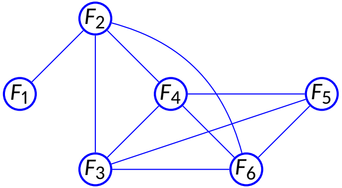
Flight	Arrv. time	Dept. time
F1	0715	0815
F2	0800	0900
F3	0830	0930
F4	0845	0945
F5	0915	1015
F6	0845	0945



Is it possible to schedule all flights using 3 gates?

Airport gate scheduling

Flight	Arrv. time	Dept. time
F1	0715	0815
F2	0800	0900
F3	0830	0930
F4	0845	0945
F5	0915	1015
F6	0845	0945

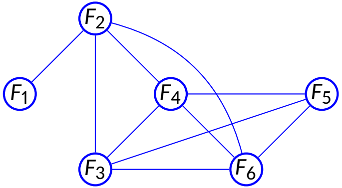


Is it possible to schedule all flights using 3 gates?

- Variables: x_i - slot for Flight F_i
- Domains: $D_i \in \{1, 2, 3\}$
- Constraints: $x_1 \neq x_2, x_1 \neq x_3, \ldots$

Airport gate scheduling

Flight	Arrv. time	Dept. time
F1	0715	0815
F2	0800	0900
F3	0830	0930
F4	0845	0945
F5	0915	1015
F6	0845	0945



Is it possible to schedule all flights using 3 gates?

- Variables: x_i - slot for Flight F_i
- Domains: $D_i \in \{1, 2, 3\}$
- Constraints: $x_1 \neq x_2, x_1 \neq x_3, \ldots$

Interval Graphs.

Cryptarithmetic

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

Cryptarithmetic

$$\begin{array}{cccccc} & S & E & N & D \\ + & M & O & R & E \\ \hline M & O & N & E & Y \end{array}$$

- Variables: $S, E, N, D, M, O, R, Y,$
- Domains: $D_i \in \{0, 1, \ldots, 9\}$
- Constraints: All different, $10 \times M + O = S + M + C_{1000}, \ldots$

Cryptarithmetic

$$\begin{array}{ccccc} & S & E & N & D \\ + & M & O & R & E \\ \hline M & O & N & E & Y \end{array}$$

- Variables: $S, E, N, D, M, O, R, Y,$
- Domains: $D_i \in \{0, 1, \ldots, 9\}$
- Constraints: All different, $10 \times M + O = S + M + C_{1000}, \ldots$

MiniZinc implementation:

```
include "alldifferent.mzn";

var 1..9: S; var 0..9: E; var 0..9: N; var 0..9: D;
var 1..9: M; var 0..9: O; var 0..9: R; var 0..9: Y;

constraint
              1000 * S + 100 * E + 10 * N + D
            + 1000 * M + 100 * O + 10 * R + E
= 10000 * M + 1000 * O + 100 * N + 10 * E + Y;

constraint alldifferent([S,E,N,D,M,O,R,Y]);

solve satisfy;
```

Knapsack

- **There are n items namely, $O_1, O_2, \ldots, O_n$. Item O_i weighs w_i and provides profit of p_i. Target is to select a subset of the items such that the total weight of the items does not exceed W and profit is maximized.**

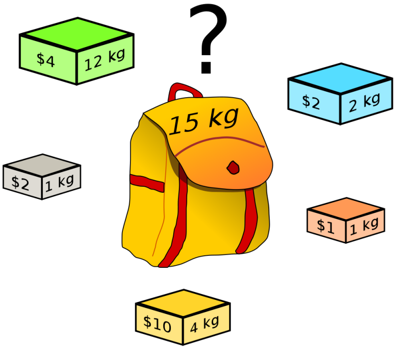


image source: Wikipedia

Knapsack

- There are n items namely, $O_1, O_2, \ldots, O_n$. Item O_i weighs w_i and provides profit of p_i. Target is to select a subset of the items such that the total weight of the items does not exceed W and profit is maximized.

  - Variables: x_i - selection of ith item
  - Domains: $\{0, 1\}$
  - Constraints: $\sum_i x_i \times w_i \leq W$
  - Optimization function: $\sum_i x_i \times p_i$

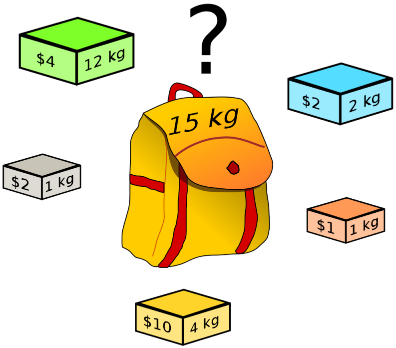


image source: Wikipedia

Warehouse planning

- **There are n possible locations to setup warehouses (W) which will deliver goods to m customers (C). Cost to setup W_j warehouse is f_j. Customer C_i has a demand of d_i which needs to fulfilled by the warehouses. Delivery cost per unit item from W_j to C_i is c_{ji}. Target is to minimize total cost to serve the required demands.**

Warehouse planning

- **There are n possible locations to setup warehouses (W) which will deliver goods to m customers (C). Cost to setup W_j warehouse is f_j. Customer C_i has a demand of d_i which needs to fulfilled by the warehouses. Delivery cost per unit item from W_j to C_i is c_{ji}. Target is to minimize total cost to serve the required demands.**
  - **Variables:** x_j **- warehouse location,** y_{ji} **- amount served by** W_j **to** C_i
  - **Domains:** $x_j \in \{0, 1\}$, $y_{ji} \in \{0, \infty\}$
  - **Constraints:** $\sum_j y_{ji} = d_i$,

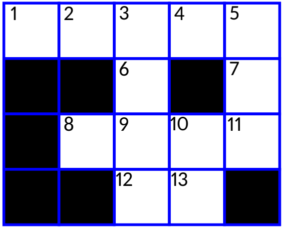
Warehouse planning

- **There are n possible locations to setup warehouses (W) which will deliver goods to m customers (C). Cost to setup W_j warehouse is f_j. Customer C_i has a demand of d_i which needs to fulfilled by the warehouses. Delivery cost per unit item from W_j to C_i is c_{ji}. Target is to minimize total cost to serve the required demands.**
  - **Variables:** x_j **- warehouse location,** y_{ji} **- amount served by** W_j **to** C_i
  - **Domains:** $x_j \in \{0, 1\}$, $y_{ji} \in \{0, \infty\}$
  - **Constraints:** $\sum_j y_{ji} = d_i$, $\sum_i y_{ji} - x_j \left(\sum_i d_i \right) \leq 0$

Warehouse planning

- **There are n possible locations to setup warehouses (W) which will deliver goods to m customers (C). Cost to setup W_j warehouse is f_j. Customer C_i has a demand of d_i which needs to fulfilled by the warehouses. Delivery cost per unit item from W_j to C_i is c_{ji}. Target is to minimize total cost to serve the required demands.**
  - **Variables:** x_j **- warehouse location,** y_{ji} **- amount served by** W_j **to** C_i
  - **Domains:** $x_j \in \{0, 1\}$, $y_{ji} \in \{0, \infty\}$
  - **Constraints:** $\sum_j y_{ji} = d_i$, $\sum_i y_{ji} - x_j \left(\sum_i d_i \right) \leq 0$
  - **Optimization function:** $\sum_i x_j \times f_j + \sum_{i,j} c_{ji} \times y_{ji}$

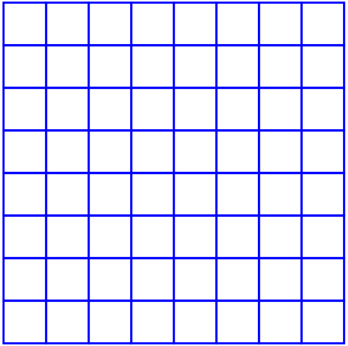
Crossword puzzle



Fill in words from the list in the given 8×8 board:
HOSES, LASER, SHEET, SNAIL, STEER, ALSO, EARN, HIKE, IRON, SAME, EAT, LET, RUN, SUN, TEN, YES, BE, IT, NO, US

- Variables: $R_1, C_3, C_5, R_8, \ldots,$
- Domains: $R_1 \in \{HOSES, LASER, SHEET, SNAIL, STEER\}, C_3 \in \{ALSO, SAME, \ldots\}$
- Constraints: $R_1[3] = C_3[1], \ldots$

Variant of crossword puzzle (practice problem)



Pack the following words in the given 8×8 board:
ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN

Rules:
- All words must read either across or down, as in a crossword puzzle.
- No letters are adjacent unless they belong to one of the given words.
- The words are rookwise connected.
- Words overlap only when one is vertical and the other is horizontal.

Solution overview

- **CSP graph creation**
  - **Create a** *node* **for** *every variable*. **All possible** *domain values* **are initially** *assigned* **to the variable**
  - **Draw** *edges* **between nodes if there is a** *binary Constraint*. **Otherwise draw a** *hyper-edge* **between nodes with constraints involving more than two variables**
- **Constraint propagation**
  - **Reduce the** *valid domains* **of each** *variable* **by applying node consistency, arc / edge Consistency, K-Consistency, till no further reduction is possible. If a solution is found or the problem found to have no consistent solution, then terminate**
- **Search for solution**
  - **Apply** *search algorithms* **to find solutions**
  - **There are interesting properties of CSP graphs which lead of efficient algorithms in some cases:** *Trees, Perfect Graphs, Interval Graphs, etc.*
  - **Issues for Search: Backtracking Scheme, Ordering of Children, Forward Checking (Look-Ahead) using Dynamic Constraint Propagation**
  - **Solving by converting to** *satisfiability (SAT)* **problems**

Search formulation of CSP

- **Standard** ***search*** **formulation of CSP**
  - **Initial state: all unassigned variables**
  - **State: partial assignment of the variables**
  - **Successor function: assign a value to unassigned variables**
  - **Goal state: all variables are assigned and satisfies all constraints**
  - **Path cost: uniform path cost**

Constraint propagation

- **Constraints**
  - **Unary constraints or node constraints (eg. $x_i \neq 9$)**
  - **Binary constraints or edge between nodes (eg. $x_i \neq x_j$)**
  - **Higher order or hyper-edge between nodes (eg. $x_1 + x_2 = x_3$)**
- **Node consistency**
  - **For every variable V_i, remove all elements of D_i that do not satisfy the unary constraints for the variable**
  - **First step is to reduce the domains using node consistency**
- **Arc consistency**
  - **For every element x_{ij} of D_i, for every edge from V_i to V_j, remove x_{ij} if it has no consistent value(s) in other domains satisfying the Constraints**
  - **Continue to iterate using arc consistency till no further reduction happens.**
- **Path consistency**
  - **For every element y_{ij} of D_i, choose a path of length L with L variables, use a consistency checking method similar to above to reduce domains if possible**

Arc consistency check (AC-3)

AC-3(*csp*) // inputs - CSP with variables, domains, constraints

queue ⟵ local variable initialized to all arcs in csp
while *queue* is not empty **do**
(X_i, X_j) ⟵ pop(queue)
if Revise(*csp*, X_i, X_j) **then**
if size of D_i = 0 **then return** *false*
for each X_k **in** X_i.neighbors-{X_j} **do**
add (X_k, X_i) to *queue*
return *true*

Revise(*csp*, X_i, X_j)

revised ⟵ *false*
for each x **in** D_i **do**
if no value y in D_j allows (x, y) to satisfy constraint between X_i and X_j **then**
delete x from D_i
revised ⟵ *true*
return *revised*

Arc consistency check (AC-3)

```
AC-3(csp) // inputs - CSP with variables, domains, constraints
    queue ← local variable initialized to all arcs in csp
    while queue is not empty do
        (X_i, X_j) ← pop(queue)
        if Revise(csp, X_i, X_j) then
            if size of D_i = 0 then return false
            for each X_k in X_i.neighbors-{X_j} do
                add (X_k, X_i) to queue
    return true
```

```
Revise(csp, X_i, X_j)
    revised ← false
    for each x in D_i do
        if no value y in D_j allows (x, y) to satisfy constraint between X_i and X_j then
            delete x from D_i
            revised ← true
    return revised
```

Complexity?

AC-3 example

- Variables: A, B, C, D
- Domain: $\{1, 2, 3\}$
- Constraints: $A \neq B$, $C < B$, $C < D$

AC-3 example

- Variables: A, B, C, D
- Domain: $\{1, 2, 3\}$
- Constraints: $A \neq B, C < B, C < D$

queue: AB, BA, BC, CB, CD, DC

AC-3 example

- Variables: A, B, C, D
- Domain: $\{1, 2, 3\}$
- Constraints: $A \neq B$, $C < B$, $C < D$

queue: AB, BA, BC, CB, CD, DC
pop(queue) // AB

AC-3 example

- Variables: A, B, C, D
- Domain: {1, 2, 3}
- Constraints: $A \neq B$, $C < B$, $C < D$

queue: AB, BA, BC, CB, CD, DC
pop(queue) // AB
No change in queue. queue=BA, BC, CB, CD, DC

AC-3 example

- Variables: A, B, C, D
- Domain: $\{1, 2, 3\}$
- Constraints: $A \neq B$, $C < B$, $C < D$

queue: AB, BA, BC, CB, CD, DC
pop(queue) // AB
No change in queue. queue=BA, BC, CB, CD, DC
pop(queue) // BA

AC-3 example

- Variables: A, B, C, D
- Domain: $\{1, 2, 3\}$
- Constraints: $A \neq B$, $C < B$, $C < D$

queue: AB, BA, BC, CB, CD, DC
pop(queue) // AB
No change in queue. queue=BA, BC, CB, CD, DC
pop(queue) // BA
No change in queue. queue=BC, CB, CD, DC

AC-3 example

- Variables: A, B, C, D
- Domain: $\{1, 2, 3\}$
- Constraints: $A \neq B$, $C < B$, $C < D$

queue: AB, BA, BC, CB, CD, DC
pop(queue) // AB
No change in queue. queue=BA, BC, CB, CD, DC
pop(queue) // BA
No change in queue. queue=BC, CB, CD, DC
pop(queue) // BC

AC-3 example

- Variables: A, B, C, D
- Domain: $\{1, 2, 3\}$
- Constraints: $A \neq B$, $C < B$, $C < D$

queue: AB, BA, BC, CB, CD, DC
pop(queue) // AB
No change in queue. queue=BA, BC, CB, CD, DC
pop(queue) // BA
No change in queue. queue=BC, CB, CD, DC
pop(queue) // BC
Remove 1. $D_B = \{2, 3\}$

AC-3 example

- Variables: A, B, C, D
- Domain: $\{1, 2, 3\}$
- Constraints: $A \neq B$, $C < B$, $C < D$

queue: AB, BA, BC, CB, CD, DC
pop(queue) // AB
No change in queue. queue=BA, BC, CB, CD, DC
pop(queue) // BA
No change in queue. queue=BC, CB, CD, DC
pop(queue) // BC
Remove 1. $D_B = \{2, 3\}$
Add AB to queue. queue=CB, CD, DC, AB
pop(queue) // CB
Remove 3. $D_C = \{1, 2\}$
No change in queue. queue=CD, DC, AB
pop(queue) // CD
No change. queue=DC, AB
pop(queue) // DC
Remove 1. $D_D = \{2, 3\}$
No change. queue=AB
pop(queue) // AB
No change in queue. queue=$\emptyset$

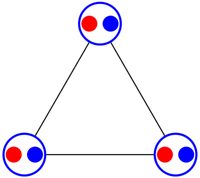
$A = \{1, 2, 3\}, B = \{2, 3\},$
$C = \{1, 2\}, D = \{2, 3\}.$

Sudoku

		3		2		6		
9			3		5			1
		1	8		6	4		
		8	1		2	9		
7					x			8
		6	7		8	2		
		2	6		9	5		
8			2		3			9
		5		1	y	3		

AC-3 limitations

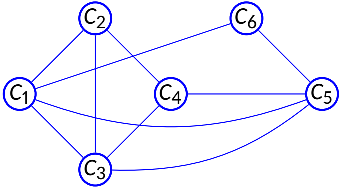
- After successful run of AC-3
  - There can be only one solution
  - There can be more than one solutions
  - There may be no solution and it fails to identify



Examination schedule

Student	Subjects
S_1	C_1, C_2, C_3
S_2	C_2, C_3, C_4
S_3	C_3, C_4
S_4	C_3, C_4, C_5
S_5	C_1, C_5, C_6

Is it possible to conduct all these exams in 3 days assuming one exam per day?



- **How does naive BFS & DFS perform?**

Backtracking search

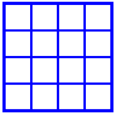
- **Backtracking is a basic search methodology for solving CSP**
- **Basic steps:**
  - **Assign one variable at a time**
    - **Fix ordering of variables (eg.**$C_1 = 1, C_2 = 3$ **is same as** $C_2 = 3, C_1 = 1$**)**
  - **Check constraint**
    - **Check with previously assigned variables**

Backtracking search

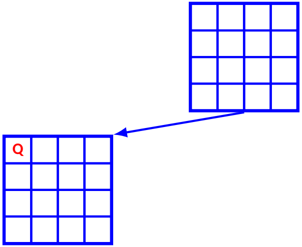
Backtrack(*assignment*)
    if *assignment* is complete **then return** *success*, *assignment*
    var ← Choose-unassigned-variable()
    for each *value* of Domain(*var*) **do**
        if *value* is consistent with the *assignment* **then**
            add *var* = *value* to *assignment*
            result = Backtrack(*assignment*)
            if *result* ≠ *failure* **return** *result*, *assignment*
    return *failure*

- **Choices:**
    - **Variable to be assigned next**
    - **Value to be assigned to the variable next**
    - **Early detection of failure**

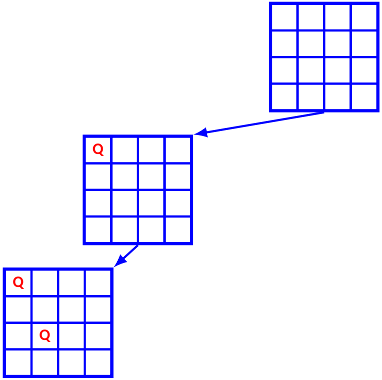
4 Queens



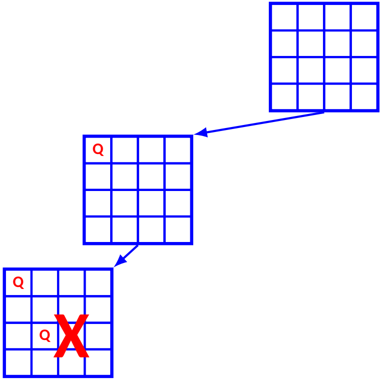
4 Queens



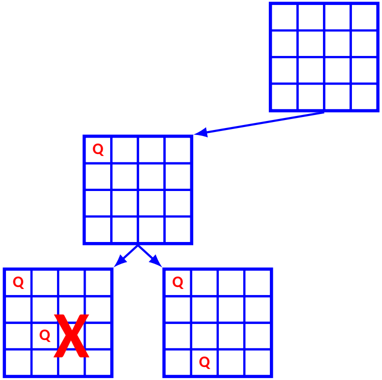
4 Queens



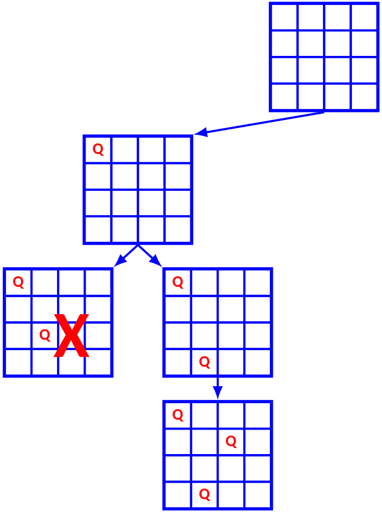
4 Queens



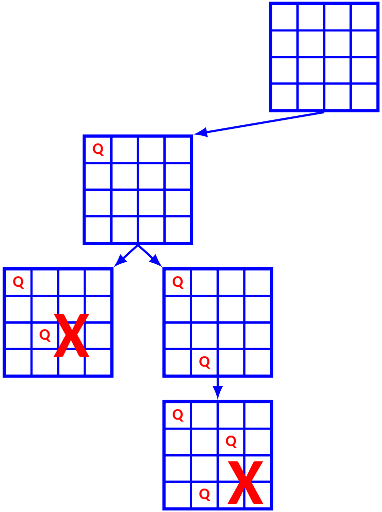
4 Queens



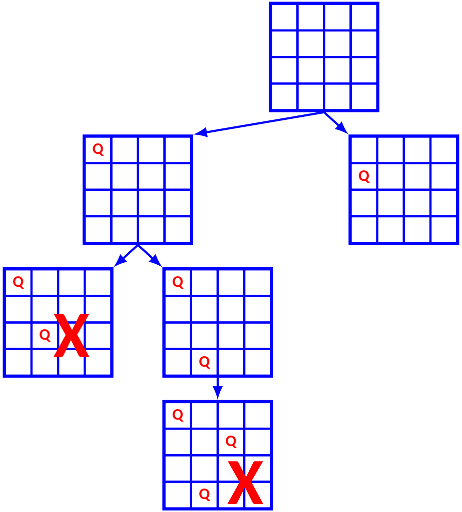
4 Queens



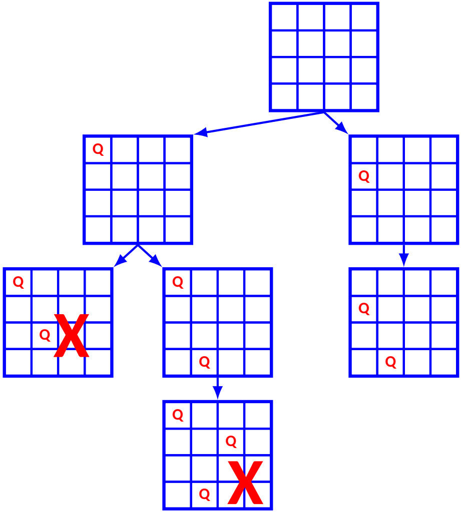
4 Queens



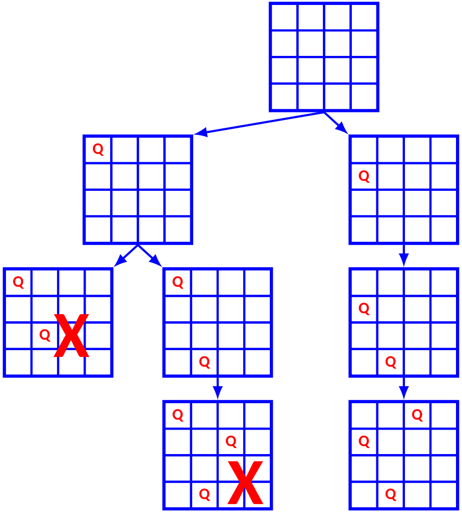
4 Queens



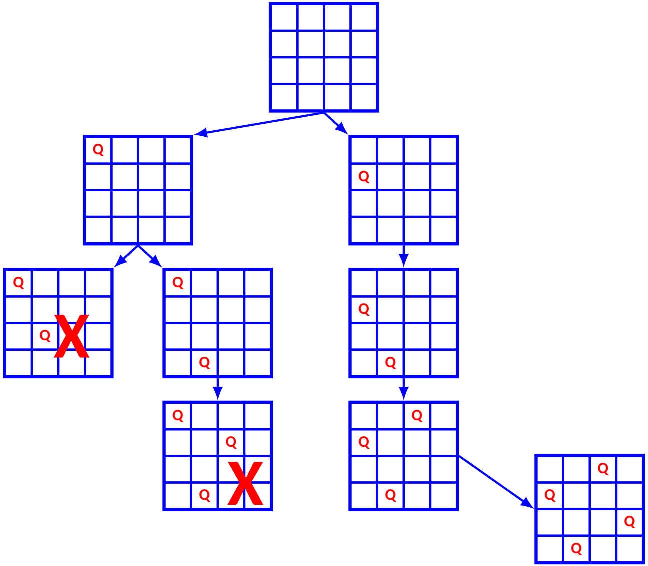
4 Queens



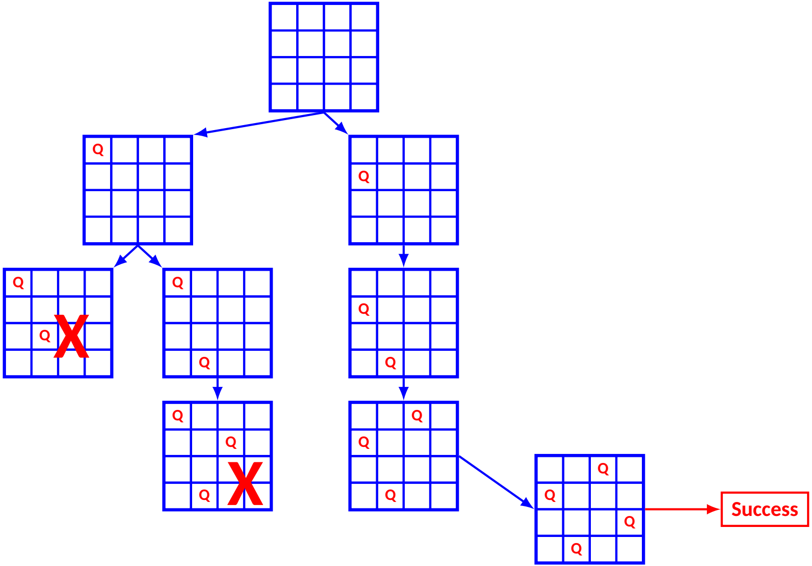
4 Queens



4 Queens



4 Queens

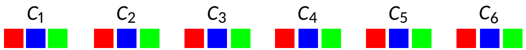


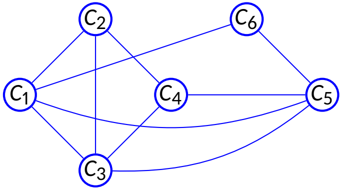
Heuristic strategy

- Variable ordering
  - Static or random
  - Minimum remaining values
    - Variable with fewest legal values (also known as most constrained variable)
  - Degree heuristic
    - Variable with the largest number of constraints on other unassigned variables
- Choice of value
  - Least constraining value
    - Value that leaves most choices for the neighboring variables in the constraint graph

Forward checking

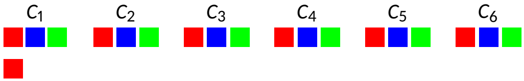
- **Forward checking propagates information from assigned to unassigned variables**

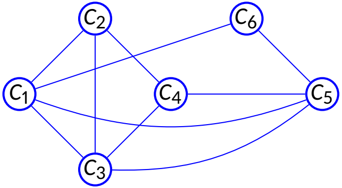




Forward checking

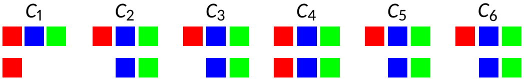
- **Forward checking propagates information from assigned to unassigned variables**

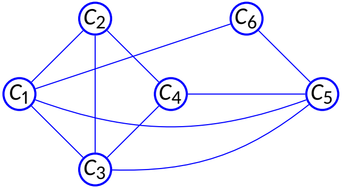




Forward checking

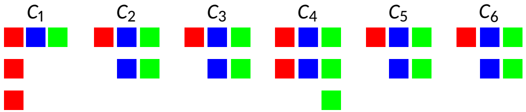
- **Forward checking propagates information from assigned to unassigned variables**

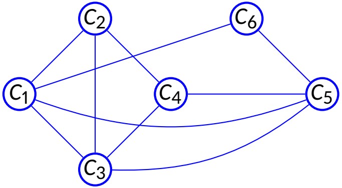




Forward checking

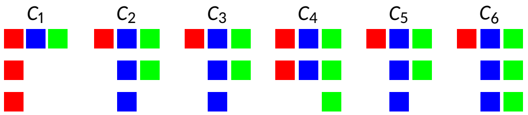
- **Forward checking propagates information from assigned to unassigned variables**

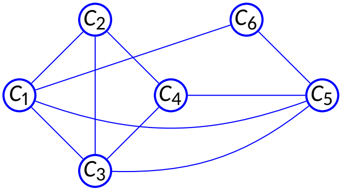




Forward checking

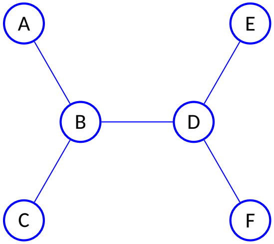
- **Forward checking propagates information from assigned to unassigned variables**





Special cases

- **General CSP problem is NP-Complete**
- **For** *perfect graphs*, *chordal graphs*, *interval graphs*, **the graph coloring problem can be solved in polynomial time**
- **Tree structured CSP can be solved in polynomial time**



Thank you!