Artificial Intelligence

Instructor: Dr. Malek Mouhoub

Department of Computer Science

University of Regina

Winter 2011

7. Constraint Processing

7.1 Introduction

"Constraint programming represents one of the closest approaches computer science has yet made to the holy grail of programming : the user states the problem, the computer solves it."

Eugene C. Freuder, Constraints, April 1997

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* V_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

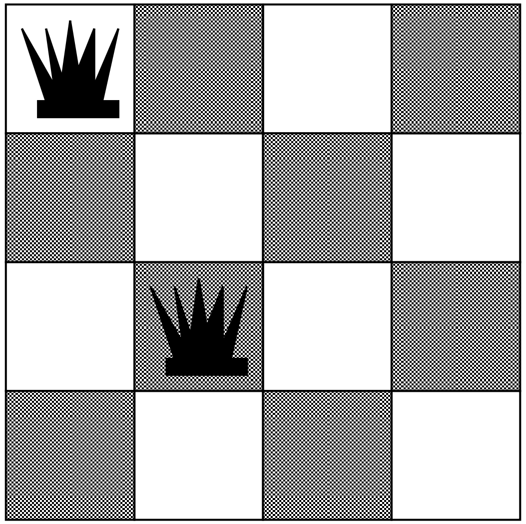
than standard search algorithms

Constraint Satisfaction Problem (CSP)

- A **Constraint Satisfaction Problem** (CSP) consist of:
  - a set of variables $X = \{x_1, \ldots, x_n\}$,
  - for each variable x_i, a finite set D_i of possible values (its domain),
  - and a set of constraints restricting the values that the variables can simultaneously take.
- A **solution to a CSP** is an assignment of a value from its domain to every variable, in such a way that every constraint is satisfied. We may want to find:
  - just one solution, with no preference as to which one,
  - all solutions,
  - an optimal, or at least a good solution, given some objective function defined in terms of some or all of the variables.
- A CSP is often represented as a (hyper)graph.

4-Queens as a CSP

Assume one queen in each column. Which row does each one go in?



Variables Q_1, Q_2, Q_3, Q_4

Domains $D_i = \{1, 2, 3, 4\}$

Constraints

$Q_i \neq Q_j$ (cannot be in same row)

$|Q_i - Q_j| \neq |i - j|$ (or same diagonal)

Translate each constraint into set of allowable values for its variables

E.g., values for (Q_1, Q_2) are $(1, 3)$ $(1, 4)$ $(2, 4)$ $(3, 1)$ $(4, 1)$ $(4, 2)$

Example: Crypt-arithmetic

Variables

$D\ E\ M\ N\ O\ R\ S\ Y$

Domains

$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

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

Constraints

$M \neq 0$, $S \neq 0$ (*unary* constraints)

$Y = D + E$ or $Y = D + E - 10$, etc.

$D \neq E$, $D \neq M$, $D \neq N$, etc.

SEND + MORE = MONEY

R_1	R_2	R_3	R_4	
	S	E	N	D
+	M	O	R	E
M	O	N	E	Y

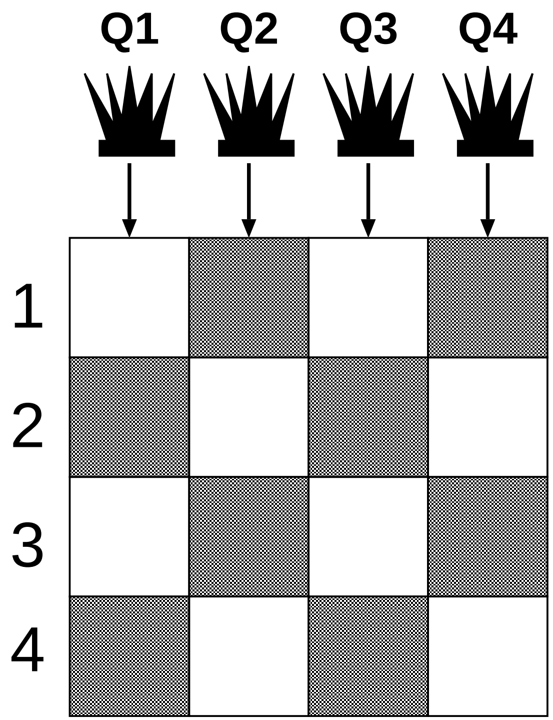
[S,E,N,D,M,O,R,Y] :: 0 . . . 9

$[R_1, R_2, R_3, R_4]$ :: 0 . . . 1

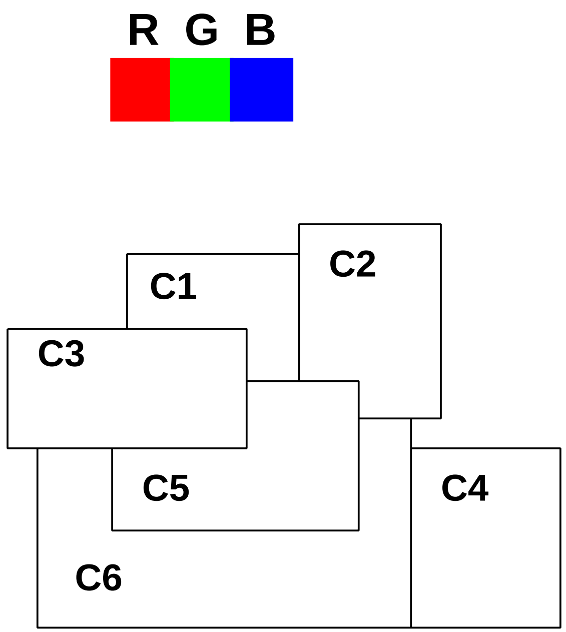
S $\neq$ 0, M $\neq$ 0

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

4-Queens and Map Coloring

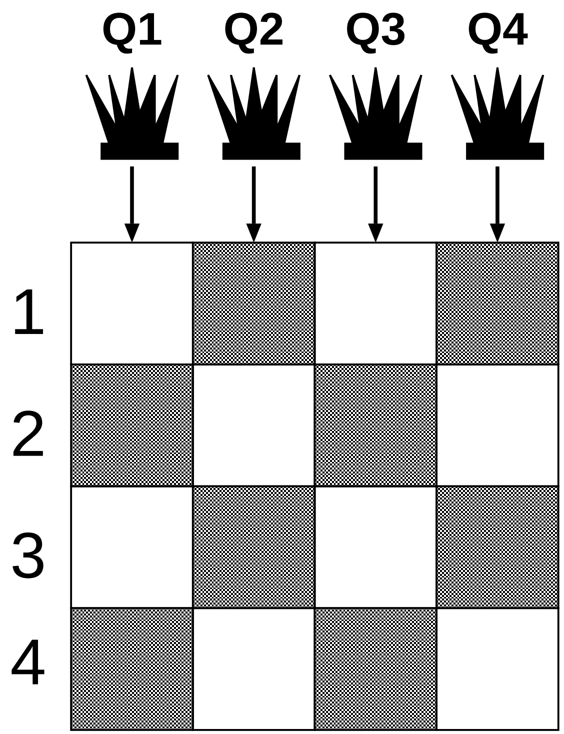


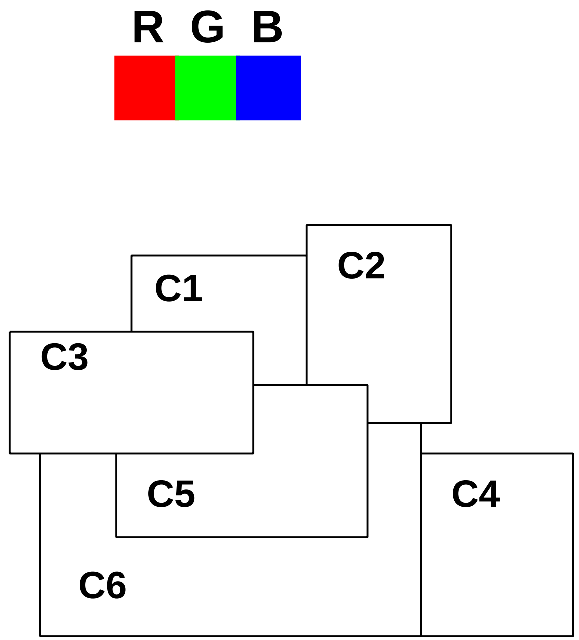
Assume one queen in each column. Which row does each one go in such that no queen constitutes an attack on any other.



Is it possible to color the map with only three colors when no two adjacent regions may share the same color

Formulation through the CSP framework

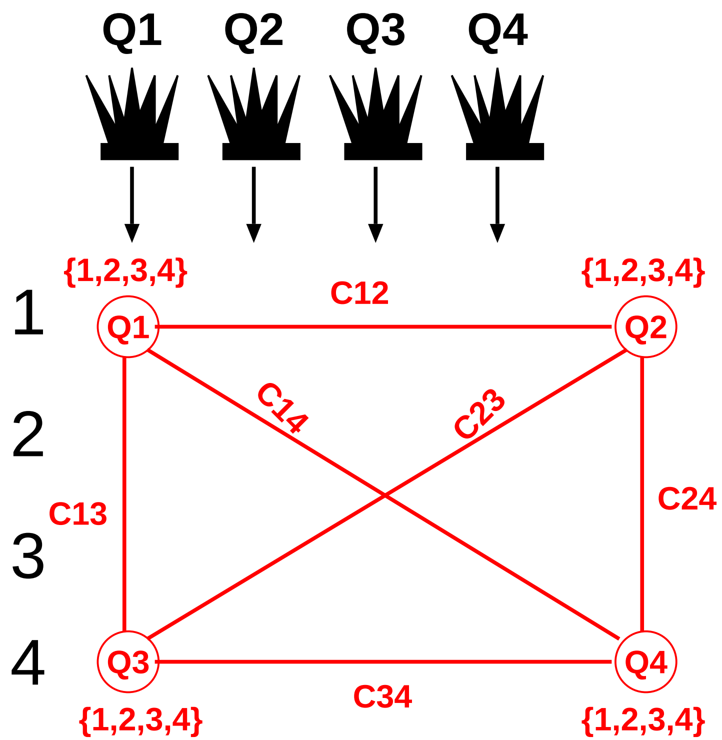


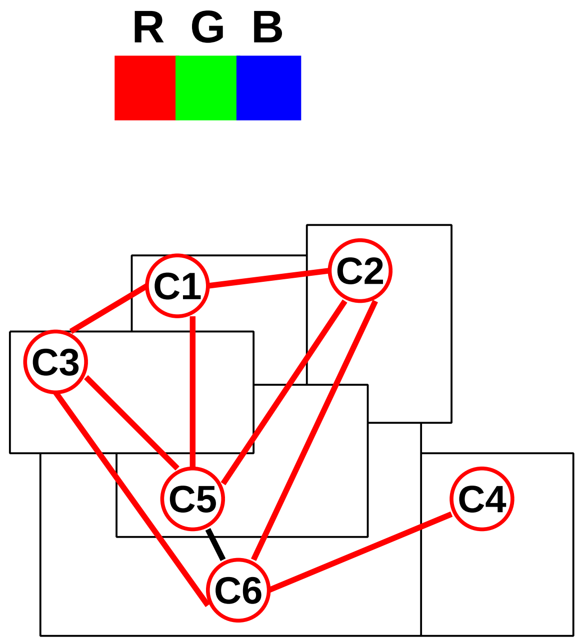


Variables: {Q1,Q2,Q3,Q4}
Domain: {1,2,3,4}
Constraints: {Qi <> Qj , |Qi-Qj| <> | i - j | }

Variables: {C1,C2,C3,C4,C5,C6}
Domain: {R,G,B}
Constraints: {C1 <> C2, C1 <> C3 ....}

Graph Representation of the CSP: Constraint Network

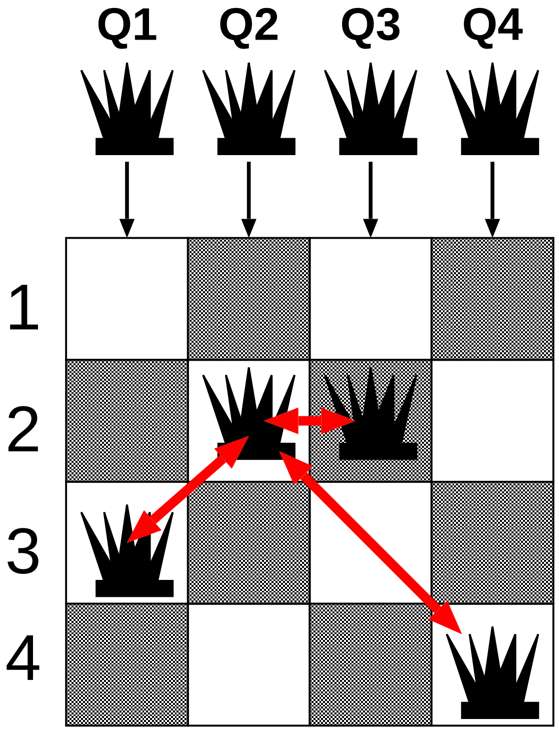


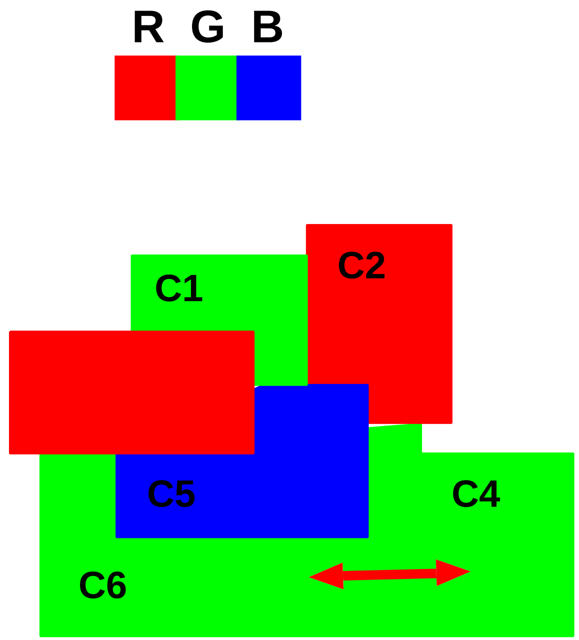


Variables: {Q1,Q2,Q3,Q4}
Domain: {1,2,3,4}
Constraints: {C12 , C13 , ... }

C12 = {(1,3),(1,4),(2,4),(3,1),(4,1),(4,2)}

Variables: {C1,C2,C3,C4,C5,C6}
Domain: {R,G,B}
Constraints: {C1 <> C2, C1 <> C3 ....}



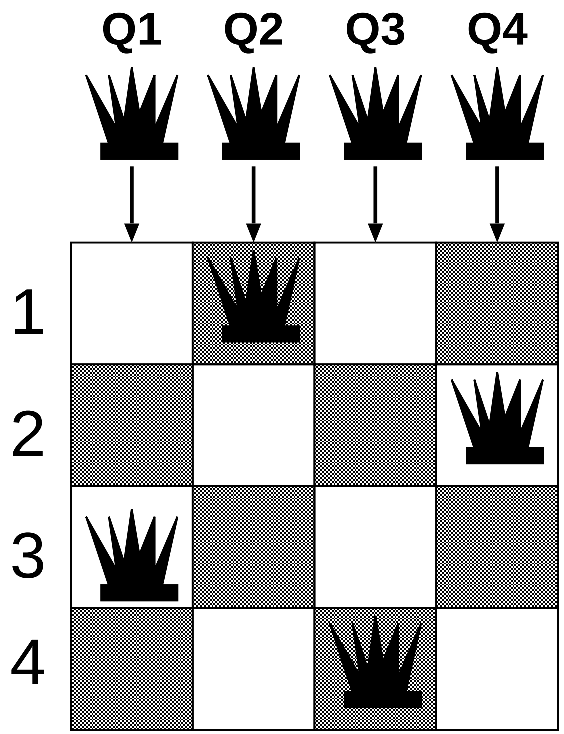


Variables: {Q1,Q2,Q3,Q4}
Domain: {1,2,3,4}
Constraints: {Qi <> Qj , |Qi-Qj| <> | i - j | }

Variables: {C1,C2,C3,C4,C5,C6}
Domain: {R,G,B}
Constraints: {C1 <> C2, C1 <> C3 ....}

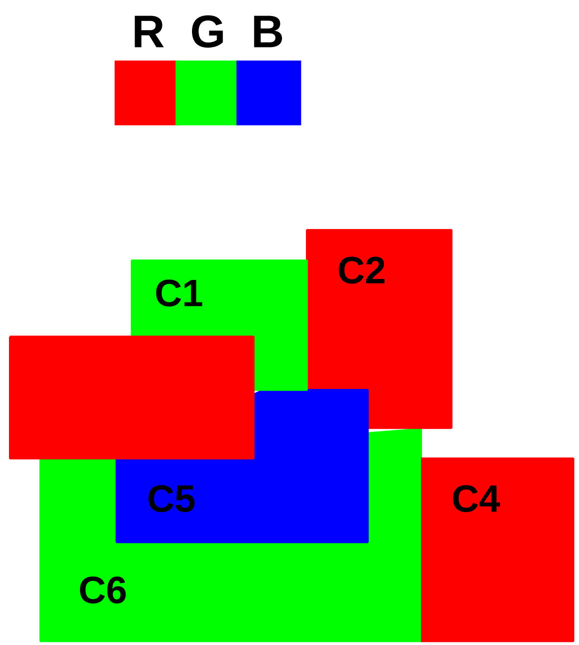
Wrong assignment !

Wrong assignment !



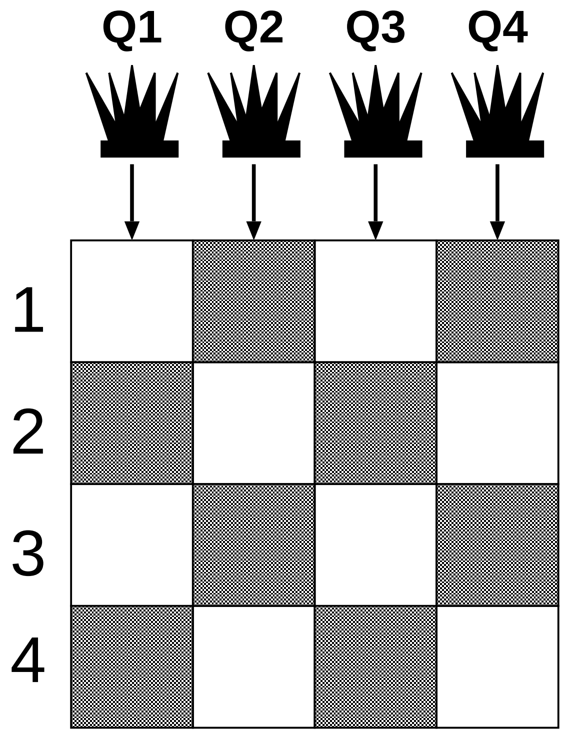
Variables: {Q1,Q2,Q3,Q4}
Domain: {1,2,3,4}
Constraints: {Qi <> Qj , |Qi-Qj| <> | i - j | }

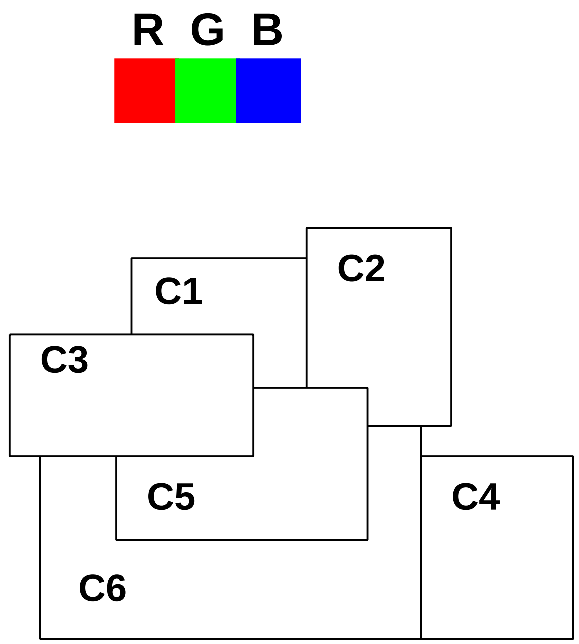
Correct assignment !



Variables: {C1,C2,C3,C4,C5,C6}
Domain: {R,G,B}
Constraints: {C1 <> C2, C1 <> C3 ....}

Correct assignment !





Variables: {Q1,Q2,Q3,Q4}
Domain: {1,2,3,4}
Constraints: {Qi <> Qj , |Qi-Qj| <> | i - j | }

4^4 = 256
complete assignments

Variables: {C1,C2,C3,C4,C5,C6}
Domain: {R,G,B}
Constraints: {C1 <> C2, C1 <> C3 ....}

3^6 = 729
complete assignments

d^n
(n: number of vars, d: domain size)

CSP is an NP-Complete Problem

Consider a CSP with n variables and d the domain size.

1. Solving the CSP requires an exponential time cost (d^n),
2. but checking to see if a complete assignment is correct can be done in polynomial time (n^c where $c \leq 2$ for binary CSPs).

CSP research work has been done on:

- Developping general algorithms for general problems: *assign values to variables and see what happens.*
  - Complete method: systematic search.
  - Incomplete method: local (or iterative) search (*trade quality for time efficiency*).
- Identifying special properties of a problem class (tractable subclass):
  - *Map coloring of the Canadian provinces.*

Current research results on CSPs work well for toy problems such as:

- N-queens,
- Zebra (five house puzzle),
- a crossword puzzle,
- cryptoarithmetics (SEND+MORE=MONEY),
- mastermind.
- Graph coloring.

Many challenges when solving real world problems such as:

- Scheduling and Planning.
- Resource allocation.
- Transportation scheduling such as crew rotering.
- Assignment problems e.g., who teaches what class.
- Timetabling problems e.g., which class is offered when and where?
- Engineering conceptual design such as hardware configuration and CAD.
- Spreadsheets and Interactive graphic : web layout.
- Molecular Biology e.g. DNA sequencing.
- Computational Linguistics.
- Temporal Databases.
- Spatial and Spatio-temporal Applications (GIS, robotics, computer games . . . etc.).
- Scene analysis.
- Network management and configuration.

What is a constraint ?

- A Constraint is an arbitrary relation over a set of variables.
  - Every variable has a set of possible values (domain).
  - The constraint restricts the possible combinations of values.
- A constraint can be described :
  - intentionally : as a mathematical/logical formula.
  - extensionally : as a table describing compatible tuples.

Example of constraints

- The circle C is inside the square S.
- The length of the word W is 10 characters.
- $X + 10 \geq Y$.
- A sum of the angles in a triangle is 180 degrees.
- The temperature in a warehouse must be in the range 0 - 5C.
- John can attend the lecture on Wednesday after 14:00.

n-ary versus binary constraints

- Many CSP algorithms are designed for binary constraints however most constraints in the real world are not binary.
- A CSP involving n-ary constraints can be transformed to an equivalent binary CSP using a transformation technique :
  - Dual encoding.
  - Hidden variable encoding.

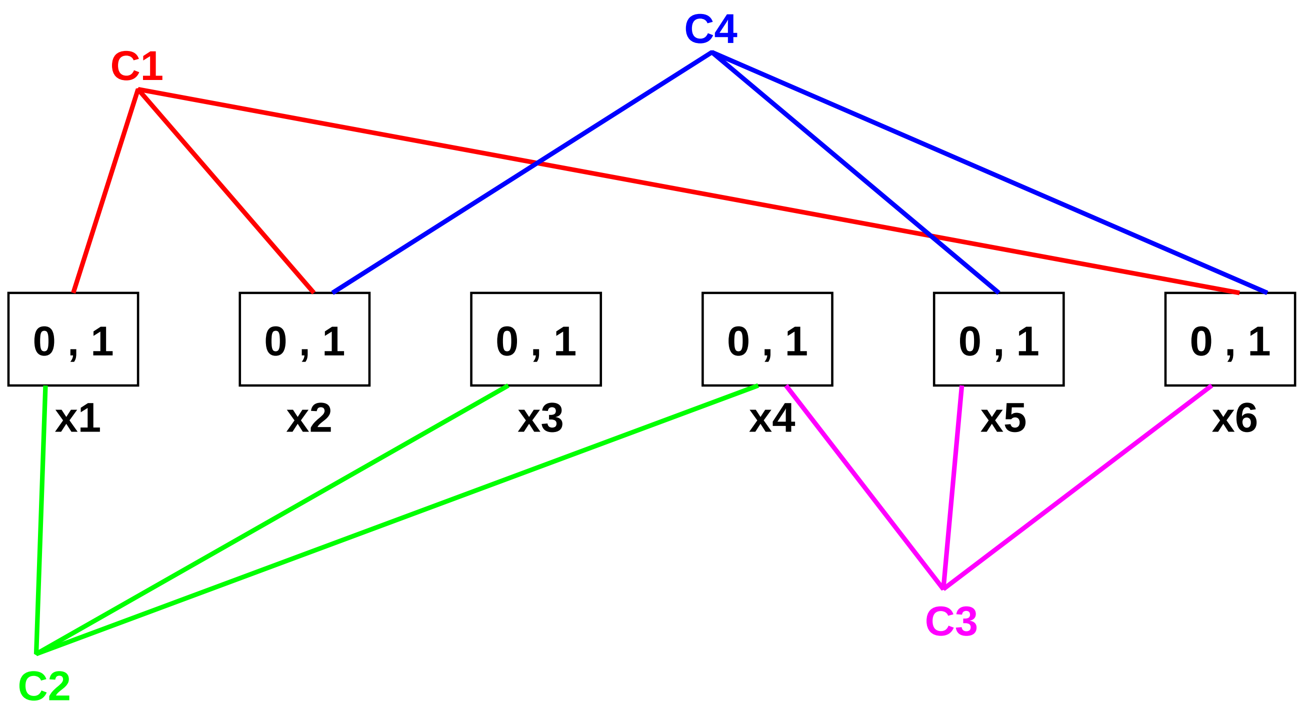
Dual encoding

- The idea consists of swapping variables and constraints.
- A n-ary constraint c is converted to a dual variable v_c with the domain consisting of compatible tuples.
- For each pair of constraints c and c' sharing some variables there is a binary constraint between v_c and v'_c restricting the dual variables to tuples in which the original shared variables take the same value.

Dual encoding

Variables: x1..x6
Domain: {0,1}
Constraints:

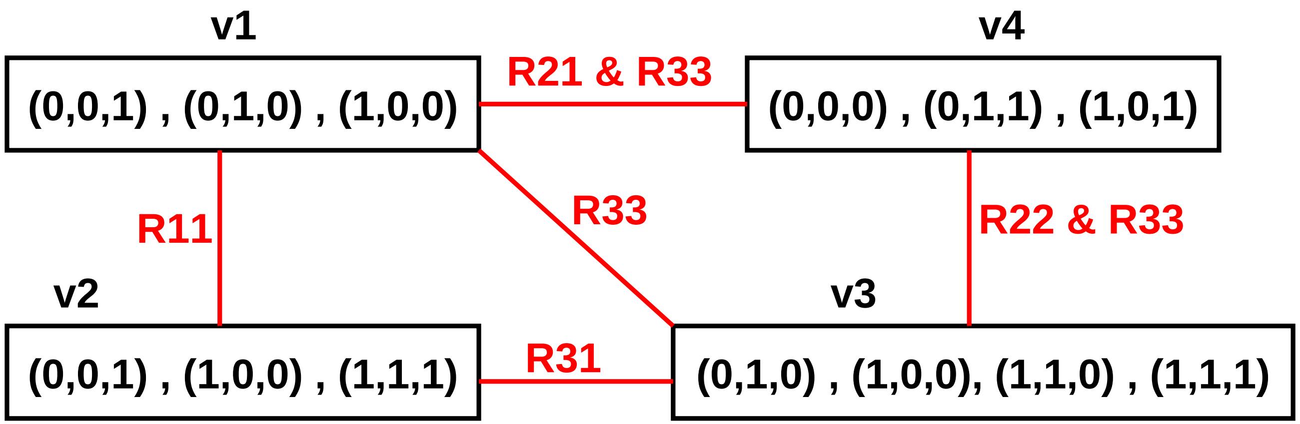
- **C1: x1+x2+x6=1**
- **C2: x1-x3+x4=1**
- **C3: x4+x5-x6>0**
- **C4: x2+x5-x6=0**



Dual encoding

Variables: x1..x6
Domain: {0,1}
Constraints:

- **C1: x1+x2+x6=1**
- **C2: x1-x3+x4=1**
- **C3: x4+x5-x6>0**
- **C4: x2+x5-x6=0**



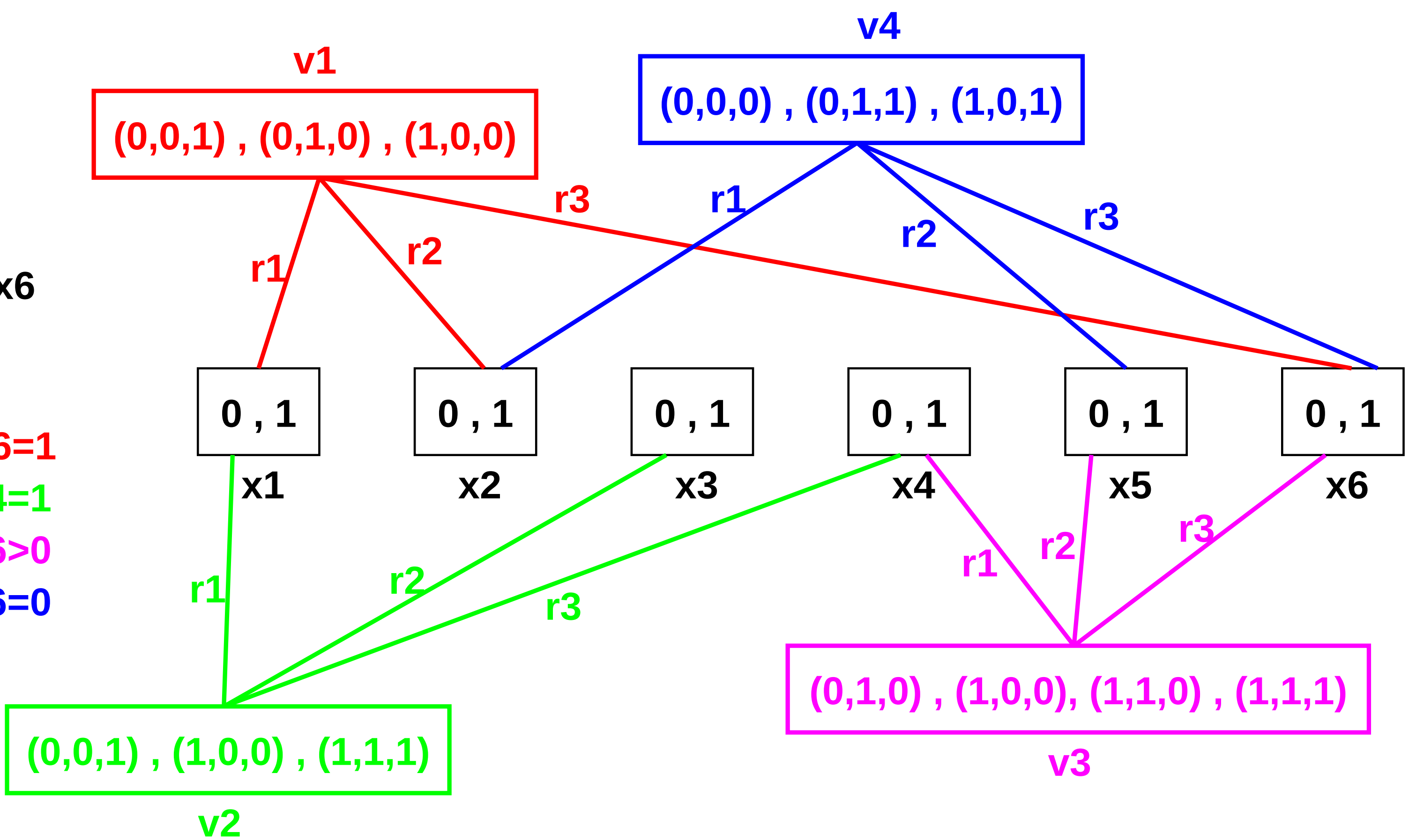
Hidden variable encoding

- New dual variables for (non-binary) constraints.
- A n-ary constraint c is converted to a dual variable v_c with the domain consisting of compatible tuples.
- For each variable x in the constraint c there is a constraint between x and v_c restricting tuples of dual variable to be compatible with x.

Hidden variable encoding

Variables: x1..x6
Domain: {0,1}
Constraints:

- **C1: x1+x2+x6=1**
- **C2: x1-x3+x4=1**
- **C3: x4+x5-x6>0**
- **C4: x2+x5-x6=0**



Graph representation of a CSP : constraint network

Scheduling Problem :

3 tasks T_1, T_2 and T_3 are processed by a mono processor machine M. A task T_4 must be processed before T_1 and T_2.

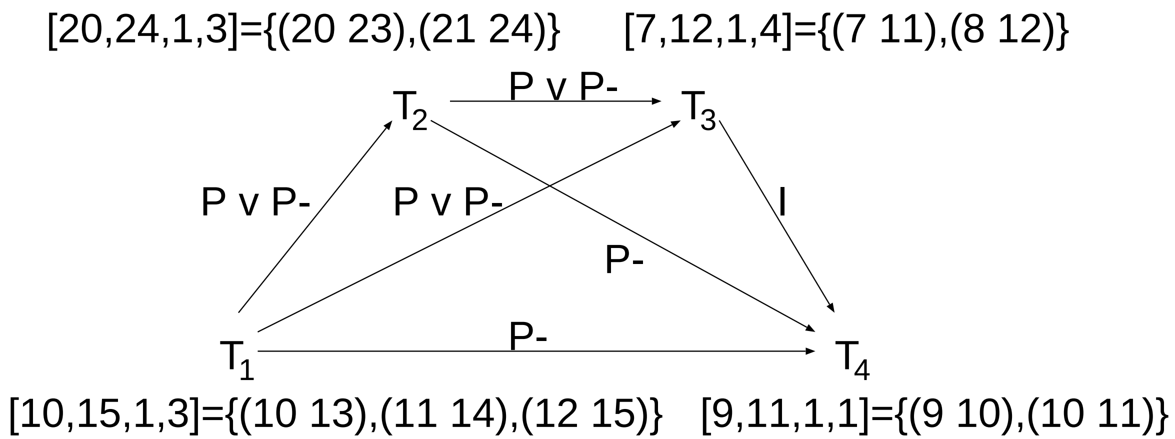
T_1 : 3h,10:00,15:00.

T_2 : 3h,20:00,24:00.

T_3 : 4h,7:00,12:00.

T_4 : 1h,9:00,11:00.

Graph representation of a CSP : constraint network



I : The universal relation(disjunction of the 13 basic Allen relations).
P : Precedes, P- : precedes inverse.

Figure 1: Scheduling problem.

7.2 Systematic Search for CSPs

Constraints are used only as a test : *assign values to variables and see what happens.*

- Systematic Search : explores the search space (space of all assignments) systematically.
- Constraint Propagation : Backtrack search algorithm preceded by and/or combined wit local consistency algorithms.
- Local search called also iterative search or non systematic search.

Systematic Search Algorithms

- Generate-and-test (GT).
- Standard Backtracking (BT).
- Backjumping (BJ).
- Dynamic Backtracking (DB).

Generate-and-test paradigm (GT)

- Systematically generates each possible value assignment and then tests to see if it satisfies all the constraints.
- The first combination that satisfies all the constraints is the solution.
- **Complexity :** $O(max(|D_i|)^n)$ where n is the number of variables.
- **Disadvantages :**
  - Generates many wrong assignments of values to variables which are rejected in the testing phase.
  - The generator leaves out the conflicting instantiations and it generates other assignments independently of the conflict.

Standard Backtracking paradigm (BT)

- Incrementally attempts to extend a partial solution toward a complete solution, by repeatedly choosing a value for another variable.

- Better efficiency than GT : as soon as all the variables relevant to a constraint are instantiated, the validity of the constraint is checked. If a partial solution violates any of the constraints, backtracking is performed to the most recently instantiated variable that still has alternatives available.

- **Complexity :** exponential for most nontrivial problems.

Standard Backtracking paradigm (BT)

- **Disadvantages :**
  - **Thrashing :** repeated failure due to the same reason. Standard backtracking algorithm does not identify the real reason of the conflict, i.e., the **conflicting variables**.
  - **Perform redundant work :** Even if the conflicting values of variables is identified during the backtrack, they are not remembered for immediate detection of the same conflict in a subsequent computation.
  - **Detects the conflict too late.**

Backjumping

- Works in a backtrack search manner and removes thrashing (skip irrelevant assignments) as follows :
  1. identify the source of conflict (impossible to assign a value)
  2. jump to the past variable in conflict
- The source of conflict (jump position) is found as follows :
  1. select the constraints containing only the currently assigned variable and the past variables
  2. select the closest variable participating in the selected constraints
- Enhancement : use only the violated constraints.

Backjumping (BJ)

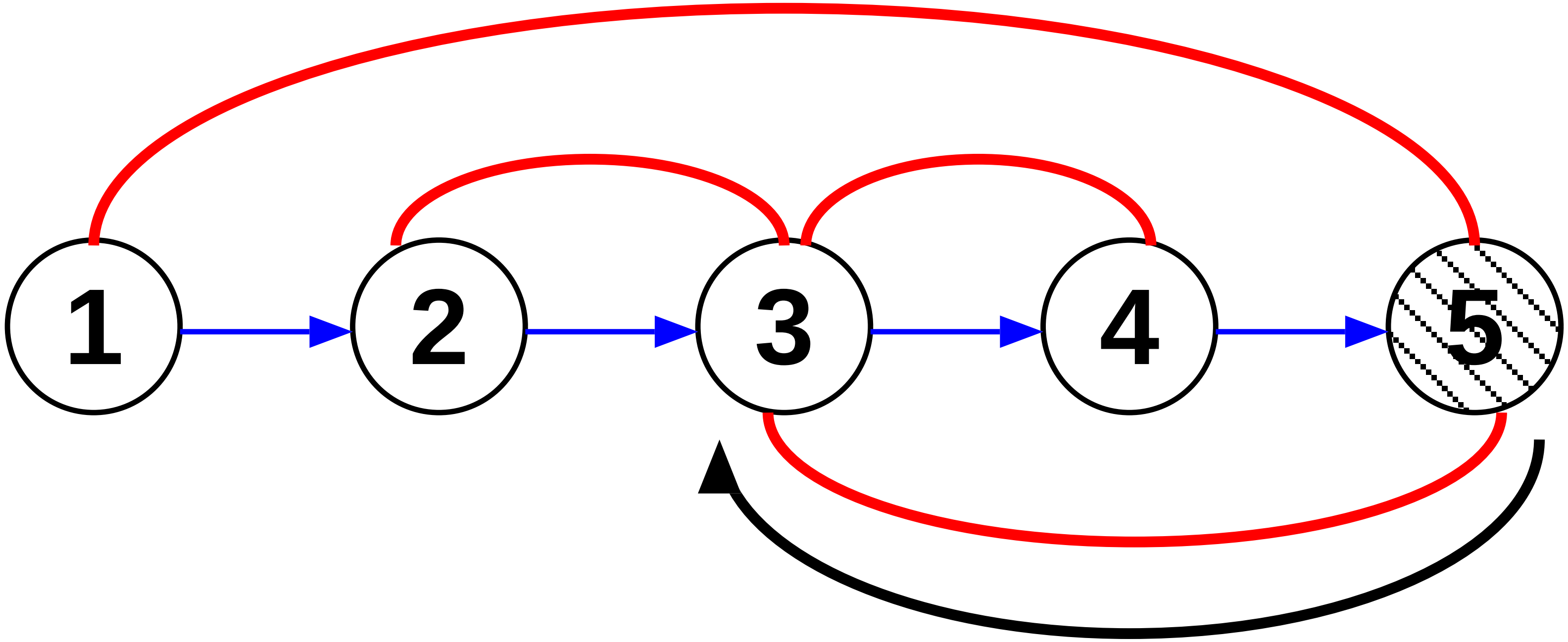
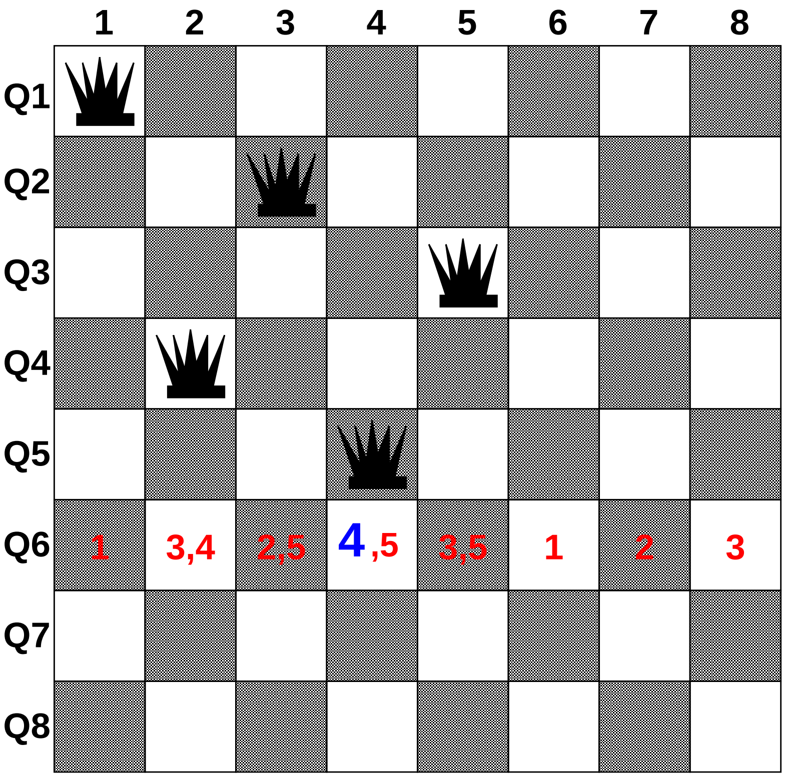


Figure 2: Graph-directed backjumping.

Conflict-directed backjumping in practice



Queens in rows are allocated to columns

6th queen cannot be allocated!

1. Write the conflicting queens to each position.

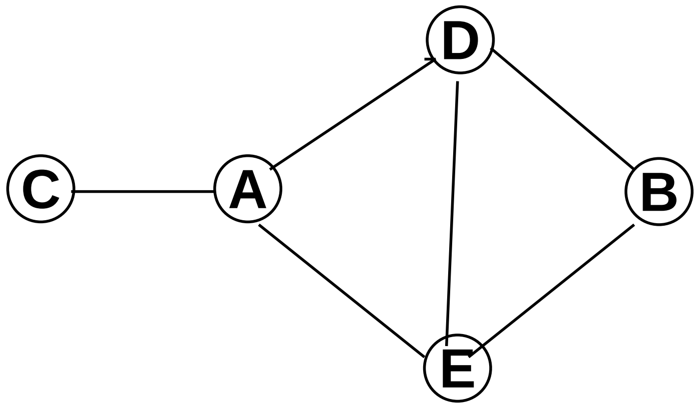
2. Select the farthest conflicting queen for each position

3. Select the closest conflicting queen among positions.

Note: Graph-directed backjumping has no effect here (due to complete grah).

Weakness of backjumping

- When jumping back the in-between assignment is lost!
- Example : colour the graph below in such a way that the connected vertices have different colours.



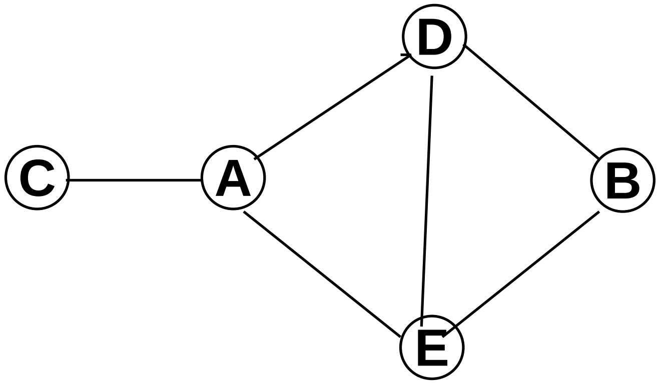
Node	Vertex		
A	1		1
B	2		1
C	1 2	BJ	1 2
D	1 2 3		1 2
E	1 2 3		1 2 3

During the second attempt to label C superfluous work is done. It is enough to leave there the original value 2, the change of B does not influence C.

Dynamic Backtracking (DB)

Dynamic Backtracking is :

- Backjumping
- + remembers the source of the conflict
- + carry the source of the conflict
- + change the order of variables



Node	1	2	3
A	X		
B		X	
C	A	X	
D	A	B	X
E	A	B	D

jump back
+ carry the conflict source

Node	1	2	3
A	X		
B		X	
C	A	X	
D	A	B	AB
E	A	B	

jump back
+ carry the conflict source
+ change the order of B,C

Node	1	2	3
A	X		
C	A	X	
B	X	A	
D	A	X	
E	A	D	X

X: selected colour
AB: a source of conflict

The vertex C (and the possible sub-graph connected to C) is not re-coloured.

7.3 Constraint Propagation

- The late detection of inconsistency is the disadvantage of GT and Backtracking paradigms.
- **A local consistency algorithm** or **consistency-enforcing algorithm** makes any partial solution of a small subnetwork extensible to some surrounding network.

  $\Rightarrow$ the inconsistency is detected as soon as possible.
- Local consistency algorithms :
  - Node consistency (1-consistency).
  - Arc consistency (2-consistency).
  - Path consistency (3-consistency).
- The backtrack search can be combined with local consistency algorithms.

Node consistency

Algorithm NC

```
for each V in nodes(G)
  for each X in the domain D of V
      if any unary constraint on V is
           inconsistent with X
       then
       delete X from D;
      endif
    endfor
  endfor
end NC
```

Arc consistency

- A graph $G = (N, R)$ (representing a constraint satisfaction problem) is arc consistent if and only if :
  $\forall i, j \in [1, n] \;\; X_i R X_j \Rightarrow \forall v_i \in D_i, \exists v_j \in D_j | (v_i, v_j) \in R$
- Arc consistency algorithms :
  - Algorithms based on arc revision : AC-1, AC-2 et AC-3[Mackworth 77].
  - Algorithms based on maintaining supports : AC-4[Mohr&Henderson86], AC-5[Deville&vanHentenryck], AC-6[Bessière94] et AC-7[Bessière95].
- arc consistency $\not\Rightarrow$ consistency of the problem ($\exists$ a solution).

Arc consistency

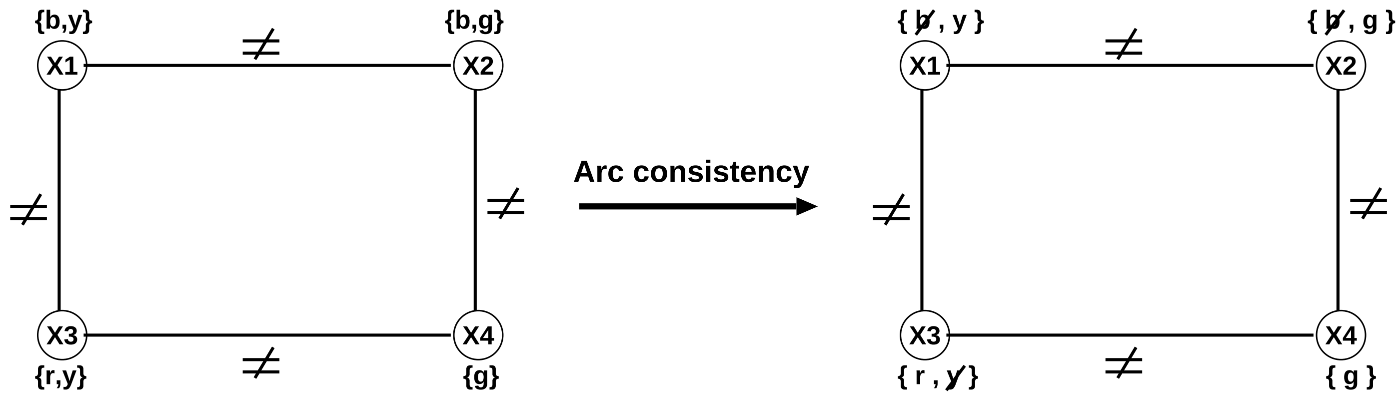


Figure 3: Performing an arc consistency algorithm.

Arc consistency

Function $REVISE(i, j)$

$REVISE \leftarrow false$

For each value $a \in D_i$ **Do**

If $\neg compatible(a, b)$ for any value $b \in D_j$ **Then**

remove a from D_i

$REVISE \leftarrow true$

End-If

End-For

Algorithm AC-3

Given a graph $G = (X, U)$

$Q \leftarrow \{(i, j) \mid (i, j) \in U\}$

(list containing all arcs of G)

While $Q \neq Nil$ **Do**

$Q \leftarrow Q - \{(i, j)\}$

If $REVISE(i, j)$ **Then**

$Q \leftarrow Q \sqcup \{(k, i) \mid (k, i) \in U \wedge k \neq j\}$

End-If

End-While

Arc consistency

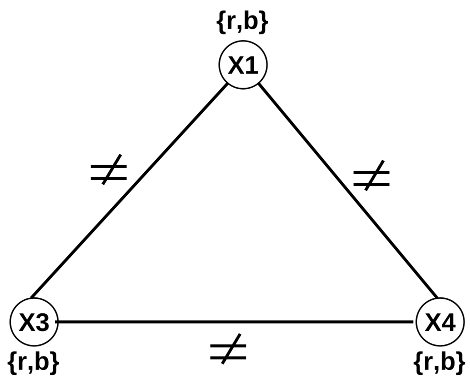


Figure 4: The problem is arc consistent but has no solution.

Path consistency

- A path $(X_0, X_1, \ldots, X_m)$ in the constraint graph for a CSP is **path-consistent** (PC) if and only if for any 2-compound label $(< X_0, V_0 >< X_n, V_n >)$ that satisfies all the constraints on X_0 and X_m there exists a label for each of the variables X_1 to X_{m-1} such that every binary constraint on the adjacent variables in the path is satisfied.
- A CSP is said to be **path consistent** if and only if every path is consistent.
- A CSP is path-consistent if and only if all paths of length 2 are path-consistent.

Path consistency

Path consistency algorithms :

- Removing the couples of values (V_i, V_j) from a relation R_{ij} if $\forall < X_k, V_k > \mid (V_i, V_k) \notin R_{ik}$ or $(V_k, V_j) \notin R_{kj}$
- PC-1, PC-2, PC-3 and PC-4.

Path consistency

Algorithm PC-2

Begin

$Q \leftarrow \{(i,k,j) \mid (i \leq j), \neq (i = k = j)\}$

(list containing all paths to check)

While $Q \neq Nil$ **Do**

$Q \leftarrow Q - \{(i,k,j)\}$

If $REVISE(i,k,j)$ **Then**

$Q \leftarrow Q \sqcup RELATED\_PATHS(i,k,j)$

End-If

End-While

End

Path consistency

Procedure REVISE(i, k, j)

Begin

$Z \leftarrow Y_{ij} \; \& \; Y_{ik} \cdot Y_{kk} \cdot Y_{kj}$

If $Z = Y_{ij}$ **Then** `return FALSE`

Else $Y_{ij} \leftarrow Z$; `Return TRUE`

End

Procedure RELATED_PATHS(i, k, j)

Begin

If $i < j$ **Then** `return`

$\{(i, j, m) | (i \leq m \leq n), (m \neq j)\} \sqcup$

$\{(m, i, j) | (1 \leq m \leq j), (m \neq i)\}$

$\sqcup \{(j, i, m) | (j \leq m \leq n)\}$

$\sqcup \{(m, j, i) | (1 \leq m \leq i)\}$

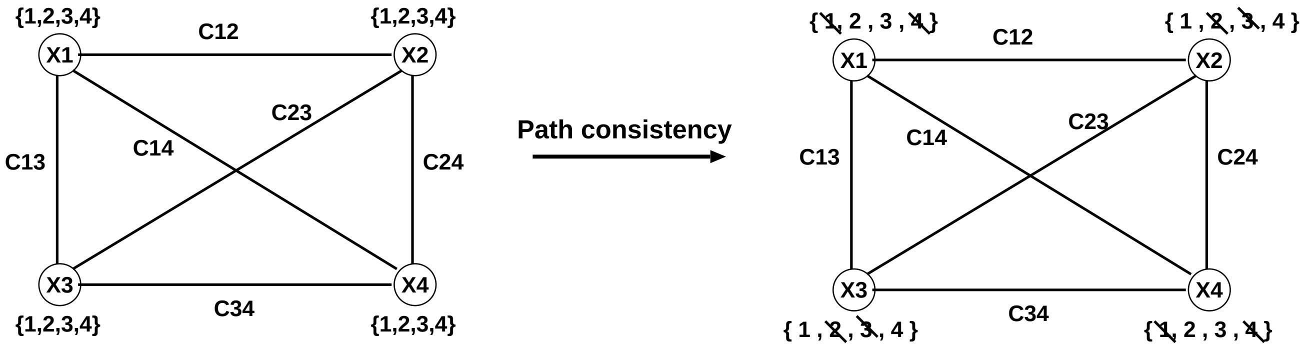
Else `Return`

$\{(p, i, m) | (1 \leq p \leq m), (1 \leq m \leq n),$

$\neq (p = i = m), \neq (p = m = k)\}$

End

Path consistency



C12 = C23 = c34 = {(1,3),(1,4),(2,4),(3,1),(4,1),(4,2)}
C13 = C24 = {(1,2),(1,4),(2,1),(2,3),(3,2),(4,1),(4,3)}
C14 = {(1,2),(1,3),(2,1),(2,3),(2,4),(3,1),((3,2),(3,4),(4,2),(4,3)}

C12 = {(2,4),(3,1)} C23 = {(1,4),(4,1)}
C13 = {(2,1),(3,4)} C24 = {(1,2),(4,3)}
C14 = {(2,3),(3,2)} C34 = {(1,3),(4,2)}

Figure 5: Applying a path consistency algorithm to the 4-queens problem .

Solution search strategies

Combine backtracking with the arc consistency algorithm.

- Backtracking.
- Forward Checking.
- Partial Look Ahead.
- Full Look ahead.

Backtracking

- Tests arc consistency among already instantiated variables.
- Detects the inconsistency as soon as it appears and, therefore, it is far away efficient than the simple generate & test approach. But it has still to perform too much search.

Backtracking

AC3-BT

Given a graph $G = (X, U)$ and a current node i
$Q \leftarrow \{(i, k) \mid (i, k) \in U \wedge k$ **already instantiated node**$\}$
    (Checking consistency between current
    and past nodes)
notconsistent $\leftarrow$ false
While $Q \neq Nil$ **and** $\neg$ notconsistent **Do**
    $Q \leftarrow Q - \{(i, j)\}$
    notconsistent $\leftarrow$ $REVISE(i, j)$
    End-If
End-While
return $\neg$ notconsistent

Backtracking

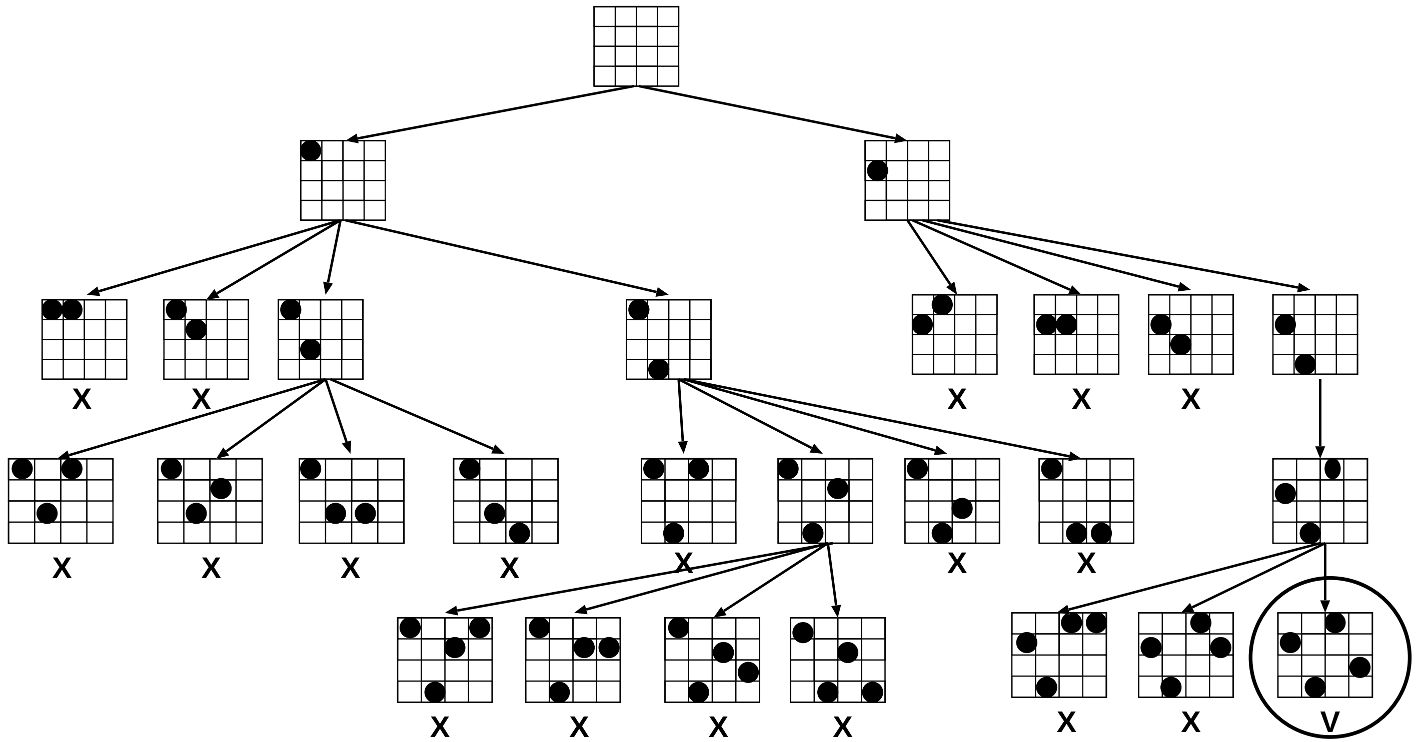


Figure 6: Applying a backtracking strategy to the 4-queens problem.

Forward Checking

- Easiest way to prevent future conflicts.
- Checks the constraints between the current variable and the future variables connecte to it via constraints.
- Allows branches of the search tree that will lead to failure to be pruned earlier than with simple backtracking.
- Whenever a new variable is considered, all its remaining values are guaranteed to be consistent with the past variables, so the checking an assignment against the past assignments is no longer necessary.

Forward Checking

AC3-FC

1\. Given a graph $G = (X, U)$ and a current node i
2\. $Q \leftarrow \{(i,k) \mid (i,k) \in U \wedge \ k$ **future node**$\}$
3\.   (checking consistency between current and
      future nodes)
4\. notconsistent $\leftarrow$ false
5\. **While** $Q \neq Nil$ **and** $\neg$ notconsistent **Do**
6\. $Q \leftarrow Q - \{(i,j)\}$
7\.   **If** $REVISE(i,j)$ **Then**
8\.   notconsistent $\leftarrow$ empty_set(D_j)
9\.   **End-If**
10\. **End-While**
11\. return $\neg$ notconsistent

Forward Checking

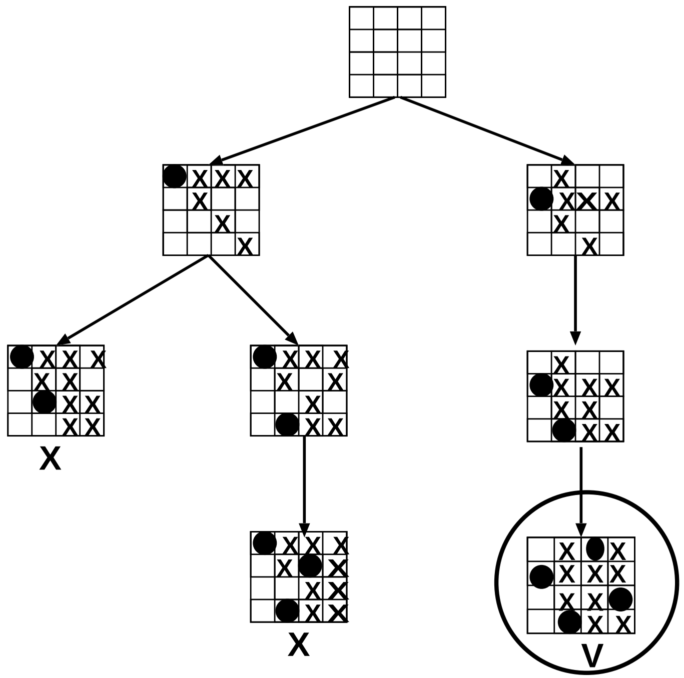


Figure 7: Applying Forward checking to the 4-queens problem

Partial Look Ahead

- Forward checking + extend the consistency checks to more future variables!
- The value assigned to the current variable can be propagated to all future variables.

Full Look Ahead

- Performs full arc consistency on the current and future nodes.
- The advantage is that it detects also the conflicts between future variables and therefore allows branches of the search tree that will lead to failure to be pruned earlier than with forward checking.
- Does even more work when each assignment is added to the current partial solution than forward checking.

Full Look Ahead

AC3-FLA

1.Given a graph $G = (X, U)$ and a current node i

2. $Q \leftarrow \{(i,k) \mid (i,k) \in U \wedge\ i, k$ **current or future node**$\}$

3. (checking consistency for current
   and future nodes)

4. notconsistent $\leftarrow$ false

5. **While** $Q \neq Nil$ **and** $\neg$ notconsistent **Do**

6. $Q \leftarrow Q - \{(i,j)\}$

7. **If** $REVISE(i,j)$ **Then**

8. $Q \leftarrow Q \sqcup \{(k,i) \mid (k,i) \in U \wedge k \neq j\}$

9. notconsistent $\leftarrow$ empty_set(D_j)

10. **End-If**

11. **End-While**

12. return $\neg$ notconsistent

Full Look Ahead

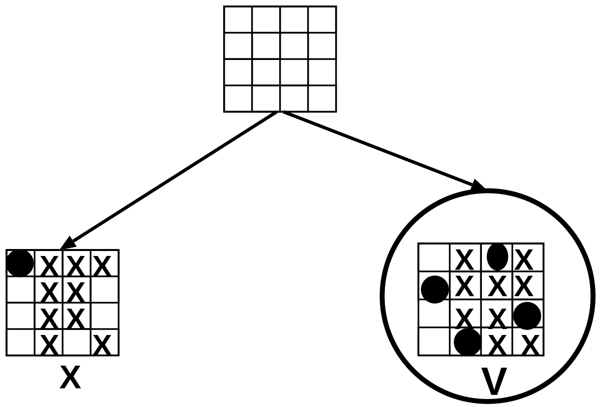


Figure 8: Applying Full Look Ahead to the 4-queens problem

Comparison of the different strategies

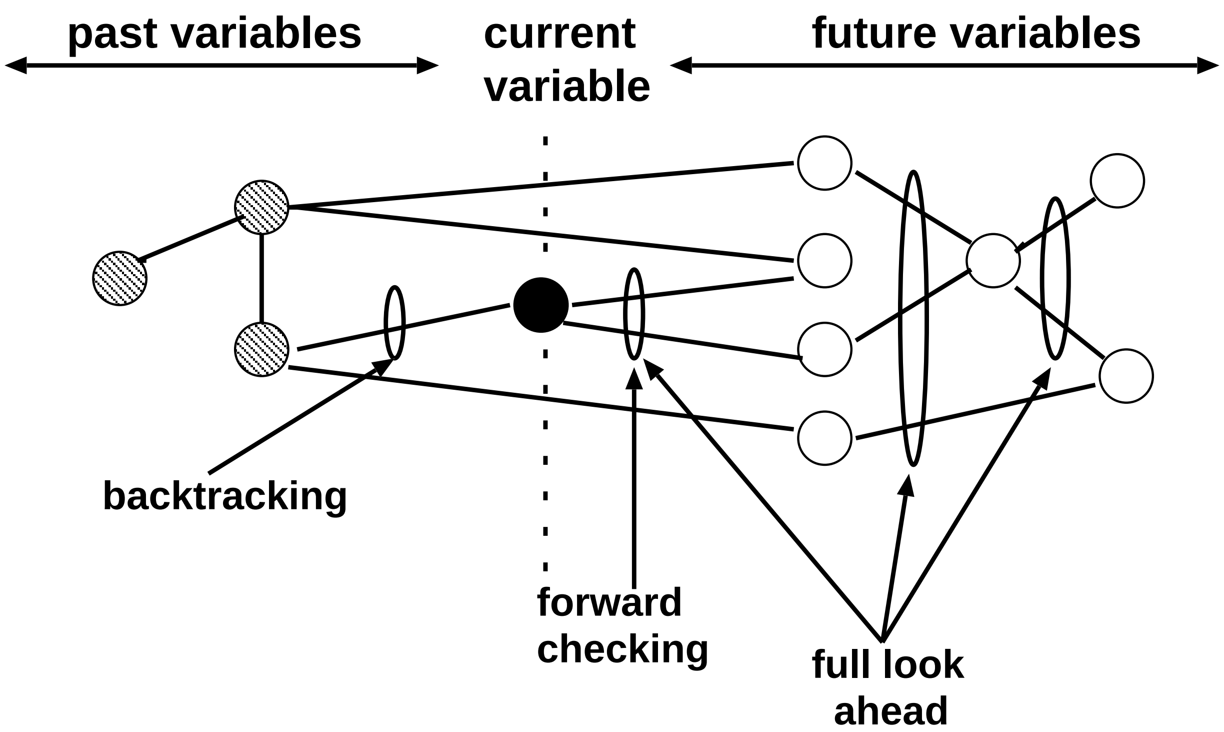


Figure 9: Comparison of the different strategies.

Comparison of the different strategies

- More constraint propagation at each node will result in the search tree containing fewer nodes,
- but the overall cost may be higher, as the processing at each node will be more expensive.
- In one extreme, obtaining strong n-consistency for the original problem would completely eliminate the need for search, but, this is usually more expensive than simple backtracking.

7.4 Heuristics for CSPs

More intelligent decisions on :

- which value to choose for each variable,
- which variable to assign next.

Given $C_1 = Red$, $C_2 = Green$, choose $C_3 =$??

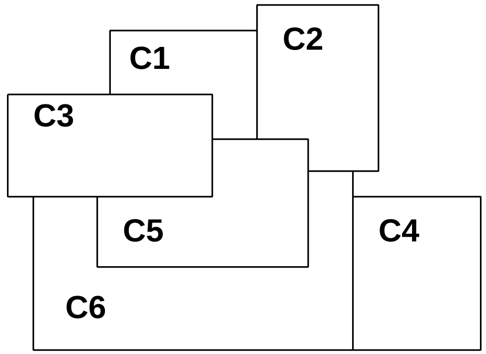
$\underline{\underline{\hspace{10em}}}$

.

Given $C_1 = Red$, $C_2 = Green$, what next??

$\underline{\underline{\hspace{10em}}}$

.



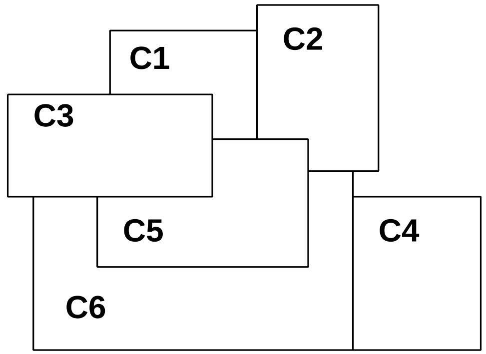
Can solve n-queens for $n \approx 1000$

Given $C_1 = Red$, $C_2 = Green$, choose $C_3 =$??

$C_3 = Green$: *least-constraining-value*

Given $C_1 = Red$, $C_2 = Green$, what next??

C_5: *most-constrained-variable*



Can solve n-queens for $n \approx 1000$

7.5 Iterative algorithms for CSPs

Hill-climbing, simulated annealing typically work with "complete" states, i.e., all variables assigned

To apply to CSPs:

allow states with unsatisfied constraints

operators *reassign* variable values

Variable selection: randomly select any conflicted variable

min-conflicts heuristic:

choose value that violates the fewest constraints

i.e., hillclimb with $h(n)$ = total number of violated constraints

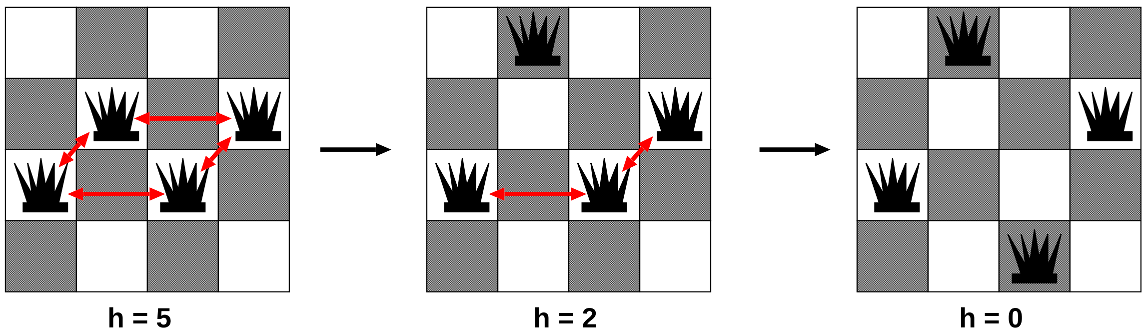
Example: 4-Queens

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

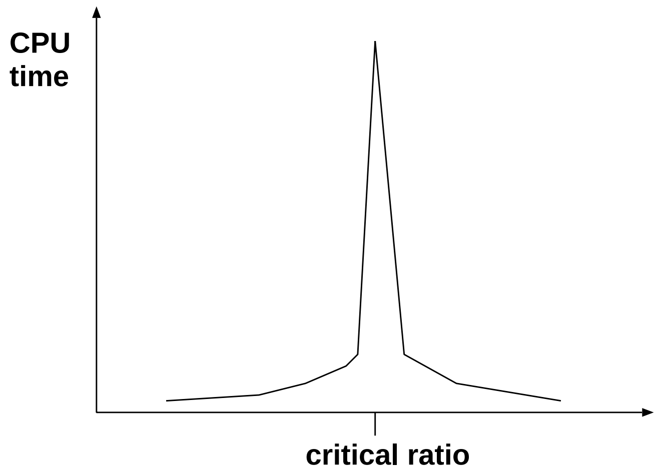


Performance of min-conflicts

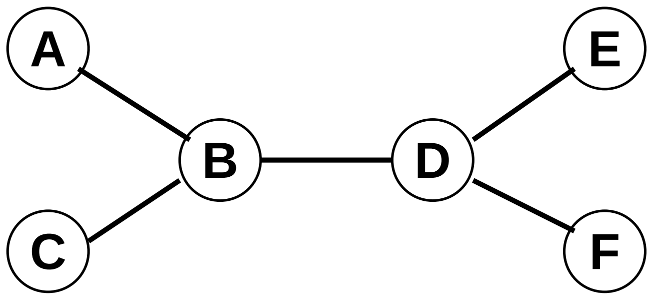
Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., n = 10,000,000)

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

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



7.6 Tree-structured CSPs



Theorem: if the constraint graph has no loops, the CSP can be solved in $O(n|D|^2)$ time

Compare to general CSPs, where worst-case time is $O(|D|^n)$

This property also applies to logical and probabilistic reasoning: an important example of the relation between syntactic restrictions and complexity of reasoning.

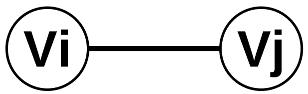
Algorithm for tree-structured CSPs

Basic step is called *filtering*:

$Filter(V_i,\, V_j)$

removes values of V_i that are inconsistent with ALL values of V_j

Filtering example:

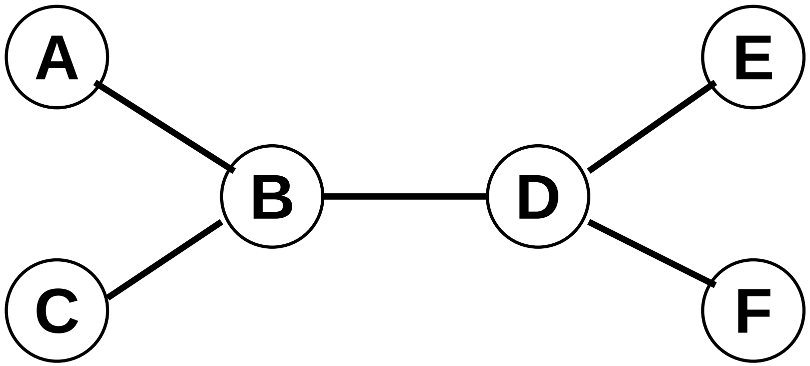


allowed pairs:

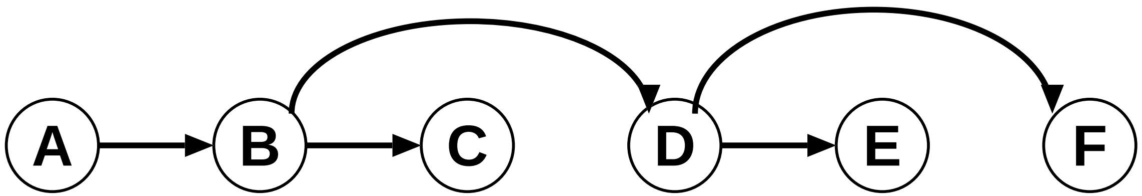
<1 , 1>
<3 , 2>
<3 , 3>

$\longrightarrow$ **remove 2 from domain of Vi**

Algorithm contd.



1) Order nodes breadth-first starting from any leaf:



2) For $j = n$ to 1, apply $Filter(V_i, V_j)$ where V_i is a parent of V_j

3) For $j = 1$ to n, pick legal value for V_j given parent value

7.7 Constraint-Based Systems

Prolog CHIP, ECLIPSe, SICStus Prolog, PROLOG IV, GNU Prolog, IF/PROLOG

C|C++ CHIP++, ILOG Solver

Java JCK, JCL, Koalog

LISP Screamer

Others Python Constraints, Mozart

Summary

CSPs are a special kind of problem:

states defined by values of a fixed set of variables

goal test defined by *constraints* on variable values

Backtracking = depth-first search with :

1. fixed variable order,
2. only legal successors.

Forward checking prevents assignments that guarantee later failure

Variable ordering and value selection heuristics help significantly

Iterative min-conflicts is <u>usually</u> effective in practice

Tree-structured CSPs can <u>always</u> be solved very efficiently