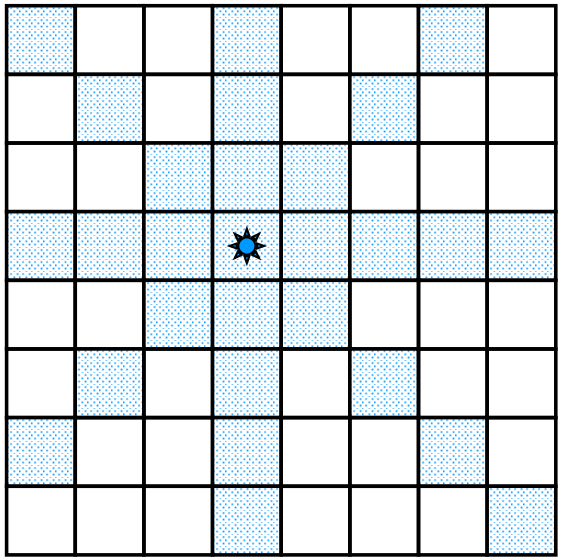
CONSTRAINT SATISFACTION

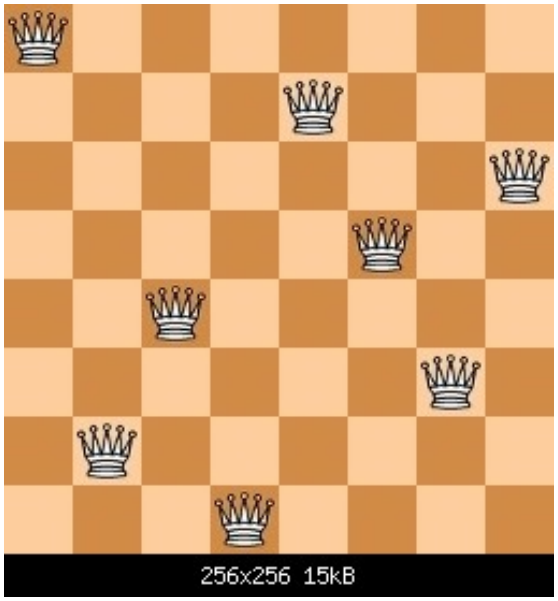
- **Many AI problems can be seen as constraint satisfaction problems.**
- **Objective: to find a state of the problem that meets a given set of constraints.**



- **Example: Eight Queens problem**
  - Given a chessboard (8x8): the problem consists in placing eight queens in order to avoid mutual attacks.
  - The possible moves for the Queen are all the positions on the same row, column and diagonal.

n QUEENS PROBLEM

- It is a mathematical problem inspired by the game of chess (1884).
- The known mathematician CF Gauss found 72 different solutions. The possible solutions are indeed 92.
- The problem can be generalized to a chessboard of *N* x *N* on which you must place N queens.
- It has been proven mathematically that for every *N* greater than 3 there exists a certain positive number of solutions; this number varies depending on *N*.



- This problem can be modelled with variables, domains and constraints and solved through search strategies.

MODELLING THE n QUEENS PROBLEM

Model 1

- The positions of the board are NXN represented by variables (8X8) x_{ij}

- The instantiation of a variable $\mathbf{x_{ij}}$ to the value **1** indicates that this position is assigned to a queen, and if the value is **0** the position is free. Domain of possible values: [0,1].

- The constraints are that there cannot be two 1 simultaneously on the same row, column or diagonal.

The constraint limit the possible combination of assignment

MODELLING THE n QUEENS PROBLEM

Model 2

- The eight queens are represented by the variables

  $x_1, x_2, \ldots, x_8$

- The subscript refers to the column occupied by the corresponding queen.

- The instantiation of a variable x_i the value k belonging to the set [1..8] indicates that the corresponding queen is placed on the k-th row of the i-th column.

- The variables have as a set of possible values for the integers between 1 and 8 which correspond to the rows occupied.

CONSTRAINTS: TWO TYPES

- Constraints: Those which bind the variables of the domain values and those who must prevent a mutual attack and which impose, then, relationships between the values assumed by the variables.

$1 \leq X_i \leq 8$	for $1 \leq i \leq 8$
$X_i \neq X_j$	for $1 \leq i < j \leq 8$
$X_i \neq X_j + (j-i)$	for $1 \leq i < j \leq 8$
$X_i \neq X_j - (j-i)$	for $1 \leq i < j \leq 8$

- The first constraint requires that the values of the variables of the problem are between the integers 1 and 8: unary constraints

- The next three constraints define relationships between the variables and, in particular, between two variables at a time: binary constraints

CONSTRAINTS: TWO TYPES

$1 \leq X_i \leq 8$	for $1 \leq i \leq 8$
$X_i \neq X_j$	for $1 \leq i < j \leq 8$
$X_i \neq X_j + (j-i)$	for $1 \leq i < j \leq 8$
$X_i \neq X_j - (j-i)$	for $1 \leq i < j \leq 8$

- The second requires that two queens are not positioned on the same line.

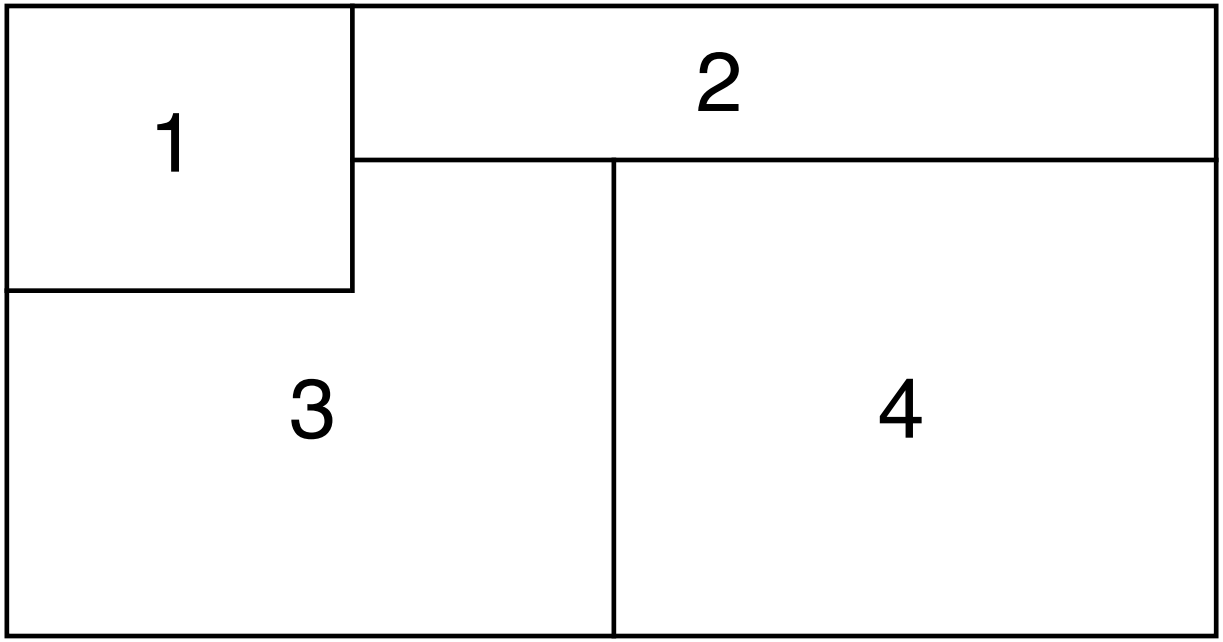
- The third and the fourth constraints concern the positions on the two diagonals starting from the initial box.

SCHEDULING AS A CSP

- Scheduling: assign tasks (with a certain duration) to resources at a given time. Resources can be shared.
- Variables: start time of activities
- Domains: possible start time of activities
- Constraints:
  - Activities can be ordered
    - Start1 + d1 <= Start2
  - The activities that use the same resource cannot overlap eg:
    - Start1 + d1 <= Start2 or Start2 + d2 <= Start1

MAP COLORING

- Suppose we need to color of portions of a map, characterized by a number, in such a way that two contiguous regions are colored with different colors. Suppose we also have the colors red (R), green (g) and blue (b).



MAP COLORING AS A CSP

- Four colors are sufficient for each map (proved in 1976). Easy Graph Coloring problem variant where at most four regions are all related.
- Variables: regions
- Domains: Colors
- Constraints:
  - adjacent regions must have different colors.

CRIPTOARITHMETICS

```
    S   E   N   D   +
    M   O   R   E   =
-------------------------
M   O   N   E   Y
```

- Objective: to determine a state in which each letter is a digit so that the initial constraints are met.

- The numbers sum as illustrated by the problem.

SUDOKU

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

SUDOKU AS A CSP

- Some boxes are already fixed, the other should be filled with numbers from 1 to 9
- The table is divided into 9 quadrants of 3x3 boxes
- Each quadrant should contain all the numbers, with no repetitions
- In addition, each horizontal row and each vertical line of the entire board must not contain repetitions
- Each box has a variable domain with from 1 to 9;
- You will see special constraints (AllDifferent) during other courses.

CSP MORE FORMALLY

- A CSP (Constraints Satisfaction Problem) is defined on a finite set of variables:
  - $(X_1, X_2, \ldots, X_n)$ decisions that we have to take
  - $(D_1, D_2, \ldots, D_n)$ domains of possible values
  - a set of constraints.

- A constraint $c(X_{i1}, X_{i2} \ldots X_{ik})$ between k variables is a subset of the cartesian product $D_{i1} \times D_{i2} \times \ldots \times D_{ik}$ that specifies which values of the variables are compatible with each other.

- This subset must not be explicitly defined but is represented in terms of relationships.

- A solution to a CSP provides an assignment of all the variables that satisfies all the constraints.

CSP MORE FORMALLY

- **CSPs can be solved through search**
- Initial state: empty assignment {}
- Successor Function: assigns a value to a variable not yet assigned
  → fail if there is none
- Goal: The assignment is complete (all variables are assigned to a feasible value).

1. Same scheme for all CSPs
2. Limited depth n if n is number of variables.
   → uses depth-first search
3. The path is irrelevant.
4. Problem with d^n leaves (if d is the cardinality of domains)

SEARCH TREE

- A possible search tree for a CSP is obtained after establishing an order for variables: each level of the tree corresponds to a variable and each node corresponds to a possible value assignment.

- Each leaf of the tree would then represent an assignment of values to all the variables. If this assignment satisfies all constraints, then the corresponding leaf represents a solution to the problem, otherwise it is a failure.

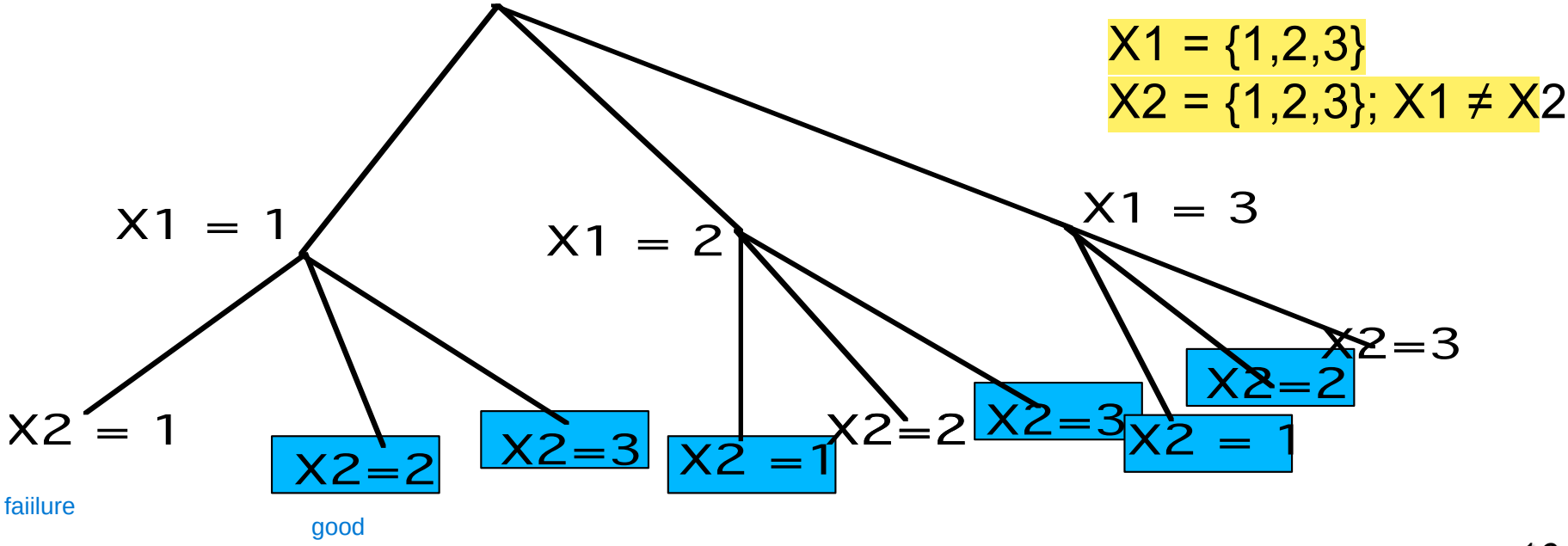
Constraints help us to reduce the size of the tree

- The search for a solution is equivalent to the exploration of the tree to find a leaf-solution.

- In an n-variable problem in which all the domains have the same cardinality d, the number of leaves of the search tree is equal to d^n.

EXAMPLE

- For example, in a tree that is a problem of 10 variables and where each domain has cardinality 10 there are 10 billion leaves.
- It is therefore evident that a smart tree exploration strategy is essential to find a solution to a complex problem in a reasonably short time (consistency techniques). Here 3^2



PROPAGATION ALGORITHMS

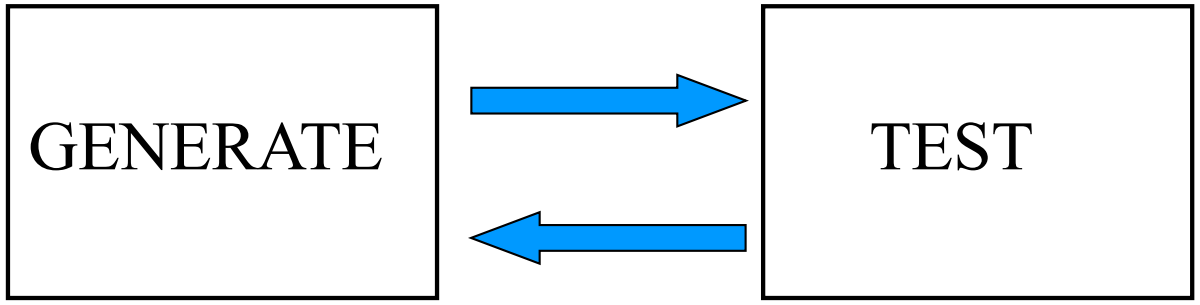
- The propagation algorithms are smart search methods that exploit constraints to prevent failures rather than recover from failures have already occurred.
- *A priori pruning* of the search tree
- Use constraints between the variables of the problem to reduce the search space before reaching the failure.
- This eliminates subtrees that lead to a failure thus limiting unnecessary backtracking.

TWO APPROACHES

- Given a CSP there are two possible approaches to its solution: one based on Consistency techniques and Propagation algorithms.
- Without loss of generality, we will refer, hereinafter, to CSP involving binary constraints (i.e constraints which involve two variables).
- **Propagation Algorithms**
  - Based on the propagation of constraints to eliminate a priori, while searching, portions of the search tree which would lead to a failure
- **Consistency Techniques**
  - Based on the propagation of constraints in order to derive a simpler problem than original.

A POSTERIORI ALGORITHMS

- The two techniques that use a posteriori constraints are:
  - The Generate and Test (GT)
  - The Standard Backtracking (SB).

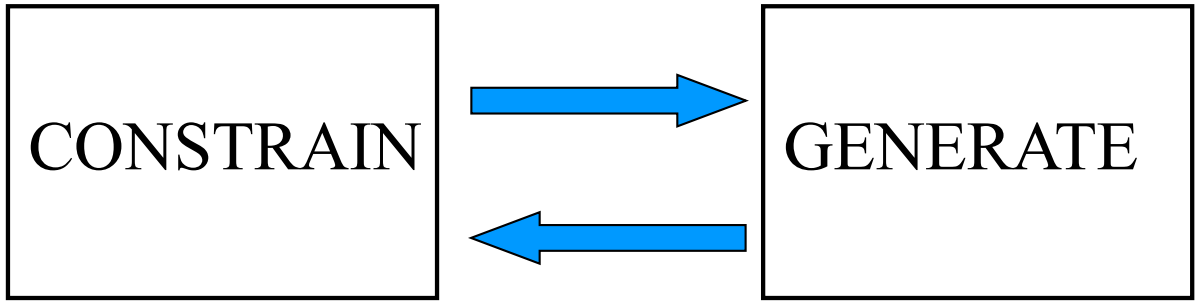


Our main goal is employ cnstraint in order t reduce the space of the tree search

PROPAGATION ALGORITHMS

we have 3 algorithms under this family

- The propagation algorithms are based on the inverse concept.
- Techniques such Forward Checking (FC), and Looking Ahead (LA).
- A module propagates the constraints as much as possible (*constrain*); at the end of propagation either we have reached solution, or a failure or new information about the free variables is required (*generated*).



TREE SEARCH

- Consider a depth-first search. It assigns a variable at a time. At each step we either:
  - Find a solution
  - Discover a failure
  - Assign another variable

- The algorithm has three degrees of freedom:
  - the choice of the variable ordering;
  - the choice of the ordering of values to be assigned to the current variable;
  - the propagation carried out in each node.

- The first two relate to search heuristics

TREE SEARCH

- The third degree of freedom is what differentiates the different strategies:

- No propagation algorithms:
  - Generate and Test
  - Backtracking Standard

- Propagation Algorithms
  - Forward Checking
  - (Partial and Full) Look Ahead

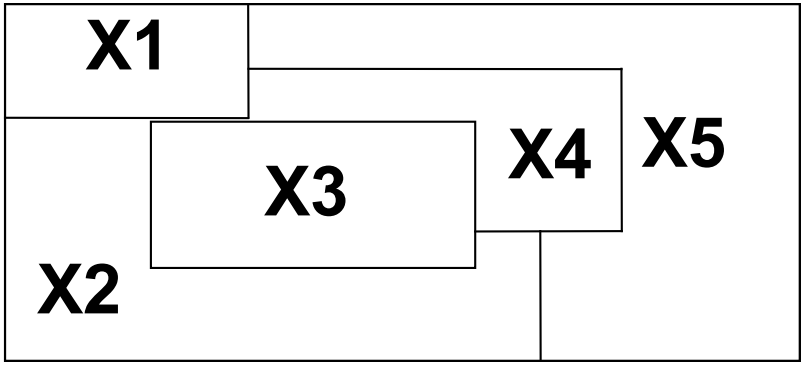
GENERATE AND TEST

- The language interpreter develops and visits a decision tree covering it in depth by assigning values to variables without bothering to verify the consistency with any constraint.
- For example, for this instance of Map Coloring problem we have:

- Domain constraints: X1, X2, X3, X4, X5 :: [red, yellow, green, blue]

- Topological Constraints:

$X1 \neq X2$, $X1 \neq X3$, $X1 \neq X4$, $X1 \neq X5$,
$X2 \neq X3$, $X2 \neq X4$, $X2 \neq X5$,
$X3 \neq X4$,
$X4 \neq X5$



- We try all permutations of colors and then check if the constraints are satisfied.
- The search tree has 4^5 leaves

try all the possible combnations of variables in the ragion and check whose is okay

But it is not good

GENERATE AND TEST FOR the 8 QUEENS

- Consider the problem of the eight queens: the variables involved in the problem require as a domain of definition, the integers between 1 and 8.

- The Generate and test assignes to the variables a permutation of the integers values in the domain

- The only constraints considered in Generate step are:
  - $1 \leq X_i \leq 8$ for $1 \leq i \leq 8$
  - $X_i \neq X_j$ to $1 \leq i < j \leq 8$
    The second is due to the fact that any attempt consists of a permutation of the values belonging to the domains, and then each value assigned to the variables is different from others.

GENERATE AND TEST

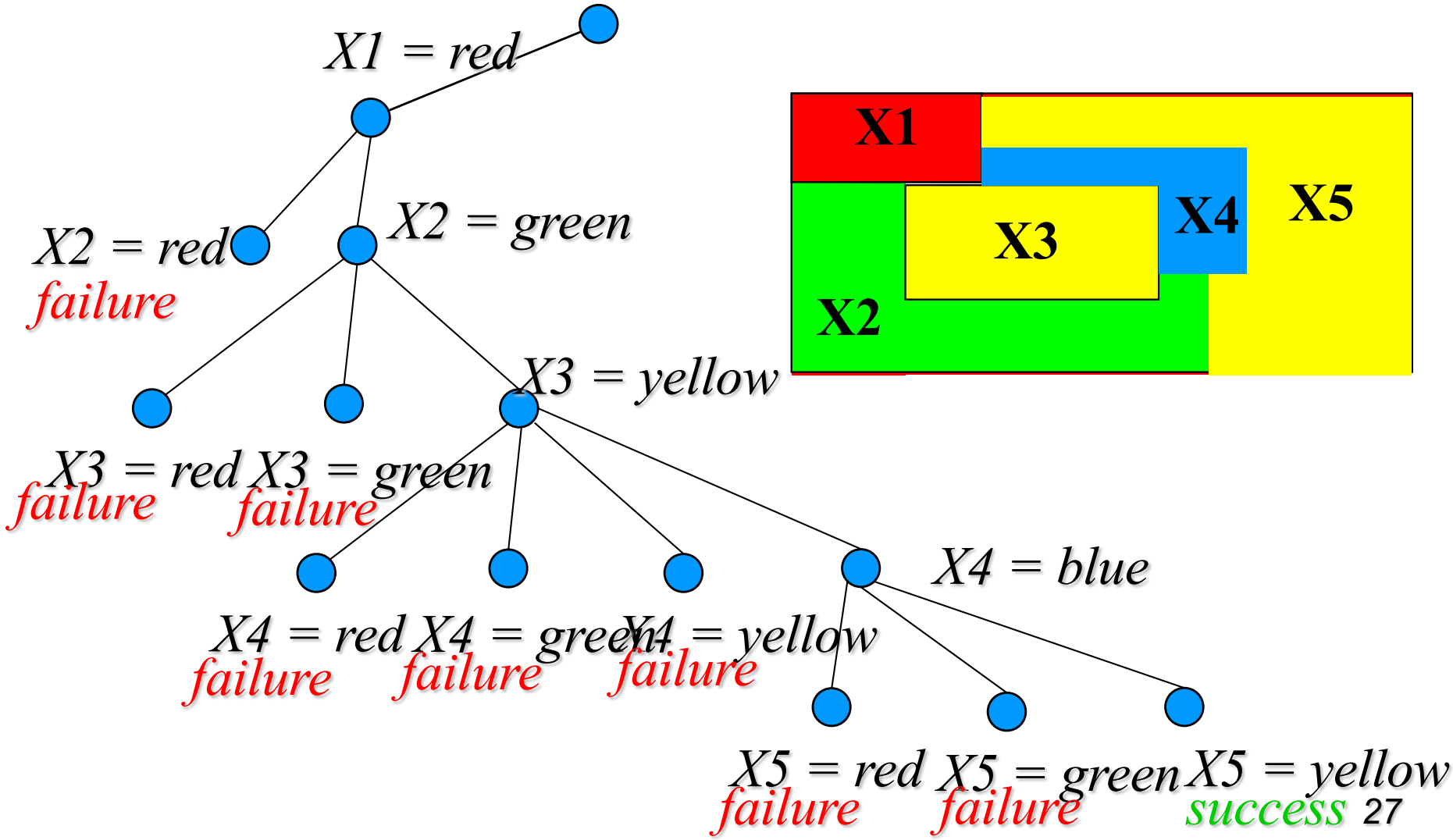
- Basic inefficiency
  - The constraints are used to limit the space of solutions after the search is performed, therefore in an *a-posteriori* fashion.
  - The number of possible permutations increases with the factorial of the number of terms to permute. If n = 8 we have a number of permutations equal to 8! = 40320, if n = 10 then n! = 3.6288 million coming for n = 20 to orders of magnitude of 10^{18} and then the size is unacceptable for a search space.

STANDARD BACKTRACKING

- Although this technique is better than the previous one, it still uses constraints in an *a-posteriori* fashion
  - At each instantiation of a variable x_i, constraints involving x_i and previously instantiated variables are checked.
  - Therefore, the use of constraints is more effective than for Generate and Test as it does not continue searching in the branches that present contradictions.

STANDARD BACKTRACKING

- Simple algorithm



STANDARD BACKTRACKING

- For the eight queens, the attempt made by Generate and Test to solve assigning all queens to a diagonal of the board would be blocked at the second instantiation.
  - In fact, the constraint $i \neq X_j - (j-i)$ for $1 \leq i < j \leq 8$ would be violated by the first two assignments replacing $X_1 = 1$ and $X_2 = 2$ in the above constraint would lead to $1 \neq 2 - (2-1)$ which leads to a contradiction

- The algorithm is then stopped and, with a backtracking, you try to assign to X_2 the value 3 with success and so on.

STANDARD BACKTRACKING

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

function RECURSIVE-BACKTRACKING(assignment, csp) returns a solution, or
failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(Variables[csp], assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment according to Constraints[csp] then
            add { var = value } to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failue then return result
            remove { var = value } from assignment
    return failure
```

STANDARD BACKTRACKING

- The constraints are used backwards (*backward*) and lead to an effective reduction of the search space w.r.t. the one produced by generate and test

- However this reduction is done backward after the assignment.

- Unlike the Generate and Test method, which leaves the constraint checking activity at the end of the instantiation of all the variables, Standard Backtracking checks their consistency at each instantiation.

STANDARD BACKTRACKING

- In the solution of the 8 queens problem suppose we have already assigned six variables to values:
  - $(X_1, X_2, X_3, X_4, X_5, X_6) = (1,3,5,7,2,4)$.
- The assignment $X_1 = 1$ is the first choice made.
- The assignment $X_2 = 1$ would violate the constraint on the column
- $X_2 = 2$ would violate the constraint on the diagonal
  - $V_{k+1} \neq V_i - (k + 1-i)$ to $1 \leq i \leq k$.
- So the assignment $X_2 = 3$ is performed that successfully satisfies all the constraints:

  $3 \neq 1$;
  $1 \leq 3 \leq 8$;
  $3 \neq 1 + (2 - 1)$;
  $3 \neq 1 - (2-1)$.
- For the third variable the value 1 violates the constraint on the row, the value 2 would violate the constraint on the diagonal with the second variable, in fact: $2 \neq 3 - (3-2)$ is not satisfied and value 3 the constraint on the row with the second variable

EXAMPLE: THE EIGHT QUEENS

- Then we proceed with X_3 =4 which violates the constraint on the diagonal with X_2
- X_3= 5 is compatible with all constraints:

  $5 \neq 1$; $5 \neq 3$; $1 \leq 5 \leq 8$;

  $5 \neq 1 + (3 - 1)$; $5 \neq 3 + (3 - 2)$;

  $5 \neq 1 - (3 - 1)$; $5 \neq 3 - (3-2)$.
- We proceed to the instantiation of the fourth variable, and all its assigments to values 1,2,3,4,5,6 violate all constraints.
- X_4= 7 is compatible with all constraints:

  $7 \neq 1$; $7 \neq 3$; $7 \neq 5$; $1 \leq 7 \leq 8$;

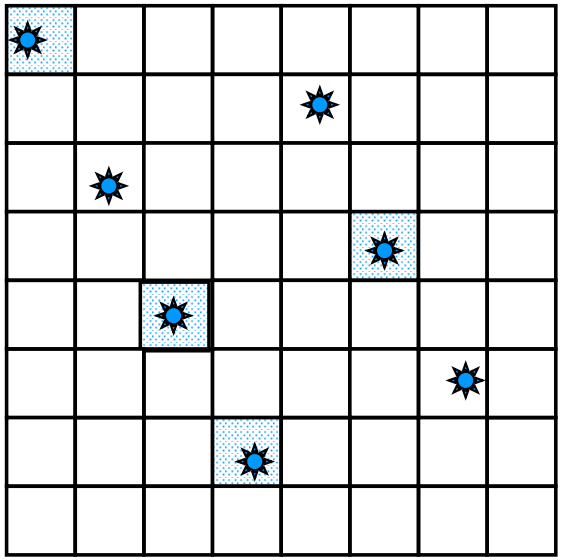
  $7 \neq 1 + (4 - 1)$; $7 \neq 3 + (4 - 2)$; $7 \neq 5 + (4-3)$

  $7 \neq 1 - (4 - 1)$; $7 \neq 3 - (4 - 2)$; $7 \neq 5 - (4-3)$
- In the same way we end up assigning X_5= 2 and X_6= 4

EXAMPLE: THE EIGHT QUEENS

For the last column, corresponding to the variable X_8 all assignments fail.

Therefore the algorithm should backtrack



LIMITATION OF STANDARD BACKTRACKING

- A pitfall of SB is that it uses constraints a-posteriori, i.e, after the instantiation is done.

  In SB before we assign and after we use the constraint

- Exploiting constraints involving free variables (still to be assigned variables) would detect this situation in advance thus avoiding expensive backtracking.

- The idea behind a priori propagation algorithms is an active use of the constraints during search. They *prune* a priori the search tree. Each variable is associated with the the set of values that are still consistent after each assignment.

- These values are stored in **domains** that are reduced during search by removing those values that would lead to a failure.

PROPAGATION ALGORITHMS

- Forward Checking (FC)
- Partial Look Ahead (PLA)
- Full Look Ahead (FLA)

- They perform increasing amount of checks on free variables.

FORWARD CHECKING

- After each assignment of variable X_i, the forward chacking propagates all constraints involving X_i. and all variables that are not yet instantiated
- This method is very effective especially when free variables domains are associated with a reduced set of allowable values and therefore are easily assignable.
  - If the domain associated to a free variable has only one value left in the domain it can be performed without any computational effort.
  - If a domain becomes empty the Forward Checking algorithm fails and backtracks
- It is based on the observation that the assignment of a value to a variable has impact on all the available values for the free variables. In this way, the constraints act forward and reduce the space of the solutions before exploring it

FORWARD CHECKING

X1 X2 X3 X4

So with this idea we will find the failure a lot before it happens

Suppose we have already assigned X1 and X2

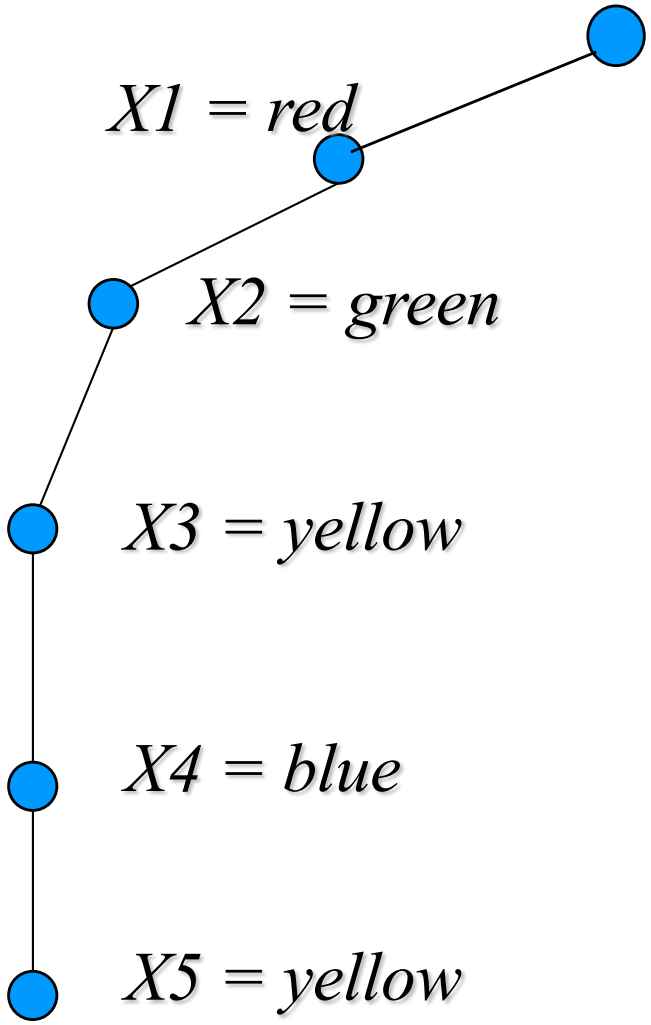
The Standard Backtracking checks c (X2, X1)

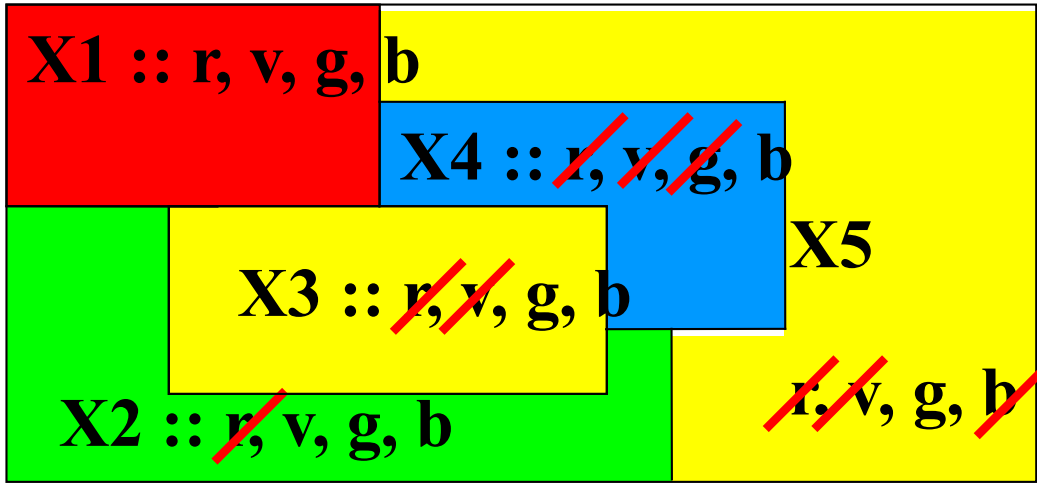
The Forward Checking checks c(X2, X3) for each value in D3
and c (X2, X4) for each value in D4
the values of D3 and D4 that are incompatible with the current assignment of X2 are deleted

If the domain of a free variable becomes empty we have a *failure*

FORWARD CHECKING

- Elimination of prior inconsistent values from variable domains of the future

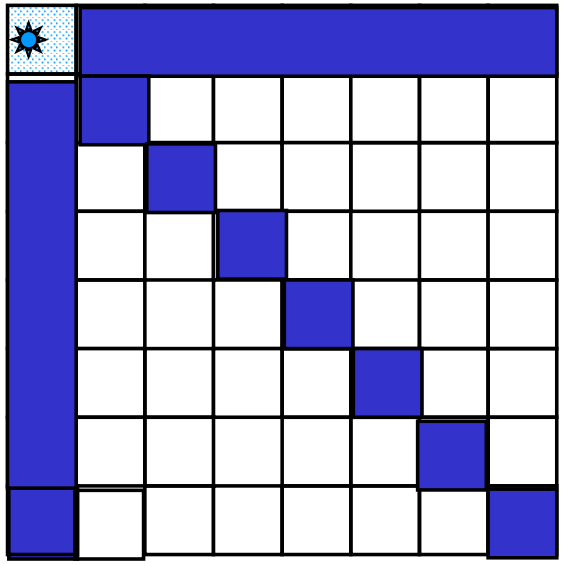




FORWARD CHECKING: 8 Queens (*cont.*)

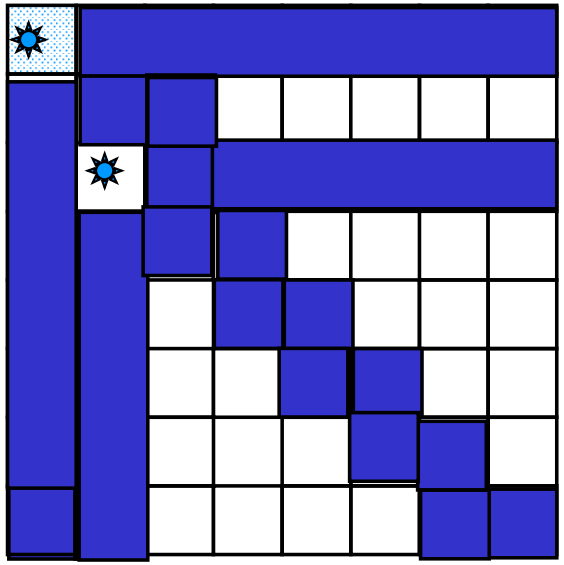
- We assign X_1=1. The FC removes value 1 from the domain of all variables and the value i from the domain of each X_i:

  X_2 is associated with the domain D_2= (3,4,5,6,7,8),
  X_3 is associated with the domain D_3= (2,4,5,6,7,8),
  X_4 is associated with the domain D_4= (2,3,5,6,7,8),
  X_5 is associated with the domain D_5= (2,3,4,6,7,8),
  X_6 is associated with the domain D_6= (2,3,4,5,7,8),
  X_7 is associated with the domain D_7= (2,3,4,5,6,8),
  X_8 is associated with the domain D_8= (2,3,4,5,6,7).



FORWARD CHECKING: 8 Queens (*cont.*)

- We thenm assign X_2=3. The domains of free variables become:
  - X_3 is associated with the domain D_3= (5,6,7,8),
  - X_4 is associated with the domain D_4= (2,6,7,8),
  - X_5 is associated with the domain D_5= (2,4,7,8),
  - X_6 is associated with the domain D_6= (2,4,5,8),
  - X_7 is associated with the domain D_7= (2,4,5,6),
  - X_8 is associated with the domain D_8= (2,4,5,6,7).



FORWARD CHECKING: 8 Queens (*cont.*)

- We assign value 5 to x_3 and propagate the constraints obtaining:
  - X_4 is associated with the domain D_4= (2,7,8),
  - X_5 is associated with the domain D_5= (2,4,8),
  - X_6 is associated with the domain D_6= (4),
  - X_7 is associated with the domain D_7= (2,4,6),
  - X_8 is associated with the domain D_8= (2,4,6,7).

The domain of variable X_6 contains only one value.

- As soon as X_4 is assigned to value 2 we obtain a failure as the domain of X_6 becomes empty:
  - X_5 is associated with the domain D_5= (4,8),
  - X_6 has domain D_6 empty,
  - X_7 is associated with the domain D_7= (4,6),
  - X_8 is associated with the domain D_8= (4,7).

FORWARD CHECKING: 8 Queens (*cont.*)

- The backtracking leads to the assignment X_4= 7.

- The algorithm narrows domains as follows:

  X_5 is associated with the domain D_5= (2,4),
  X_6 is associated with the domain D_6= (4),
  X_7 is associated with the domain D_7= (2,6),
  X_8 is associated with the domain D_8= (2,4,6).

- Then X_5=2 leads to

  X_6 is associated with the domain D_6= (4),
  X_7 is associated with the domain D_7= (6),
  X_8 is associated with the domain D_8= (4,6).

- When assigning the two singleton variables we obtain a failure
- And backtrack again

FORWARD CHECKING: 8 Queens (*cont.*)

- The backtracking leads to the assignment X_4= 7.

- The algorithm narrows domains as follows:
  X_5 is associated with the domain D_5= (2,4),
  X_6 is associated with the domain D_6= (4),
  X_7 is associated with the domain D_7= (2,6),
  X_8 is associated with the domain D_8= (2,4,6).

Note that if we reason on top of unassigned variables we can already infer something

- Then X_5=2 leads to
  X_6 is associated with the domain D_6= (4),
  X_7 is associated with the domain D_7= (6),
  X_8 is associated with the domain D_8= (4,6).

- When assigning the two singleton variables we obtain a failure
- And backtrack again

LOOK AHEAD

They look constraint between pair of future variable

- Beside checking the constraints with the current instantiated variable, look ahead also checks the non assigned variables

- L*ook Ahead* checks the existence, in the domains associated with the non assigned variables, of values that are compatible with the constraints containing only non-instantiated variables.

- It is verified the possibility of a future consistent assignment between (pairs of) free variables.

In forward checking we are taking in account Xi with te future

Look ahead checks pair future variable

LOOK AHEAD

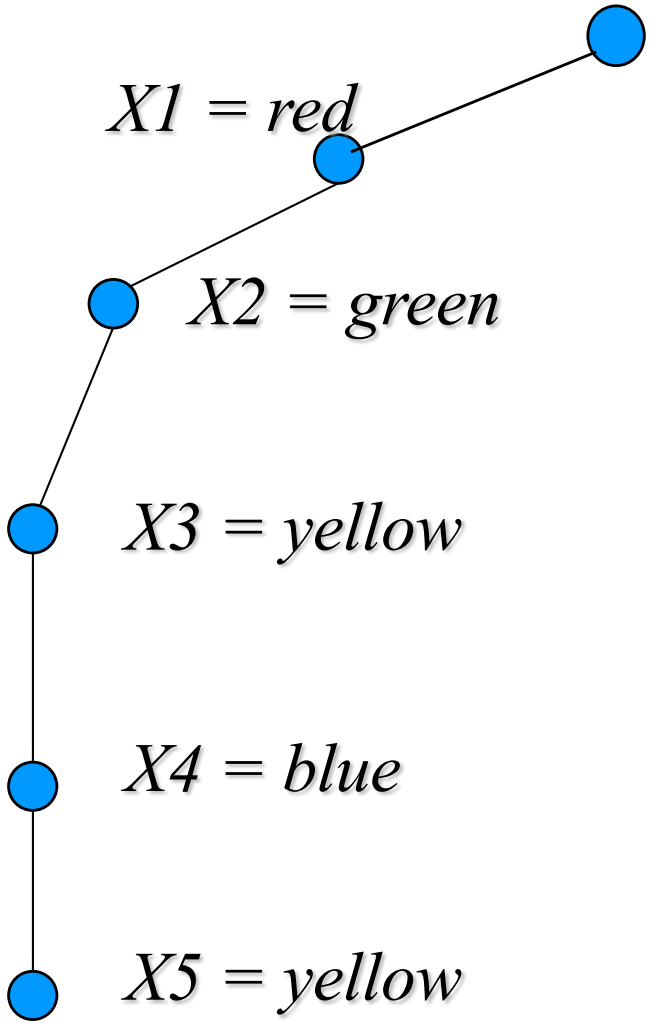
- Partial Look Ahead (PLA) and Full Look Ahead (FLA)
- Suppose we have assigned variables from X_1 to X_{h-1}

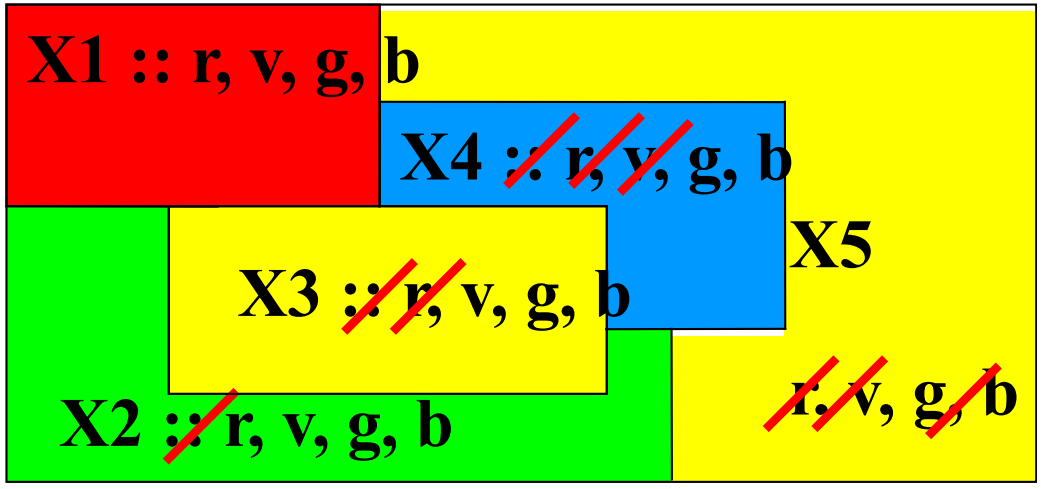
- PLA: constraint propagation containing the not yet assigned variable X_h with not yet instantiated "future" variables, ie variables X_{h+1}, ..., X_n
  - For each value in the domain of X_h it checks if in the domain of not yet assigned variable X_{h+1}, ..., X_n there is a value compatible with it.

- FLA: constraint propagation containing the not yet assigned variable X_h with not yet instantiated variables X_{k+1}... X_{h-1}, X_{h+1}.., X_n.
  - For each value in the domain of X_h it checks if in the domain of not yet assigned variable X_{k+1}... X_{h-1}, X_{h+1}.., X_n there is a value compatible with it.

FC vs PLA for Map Coloring

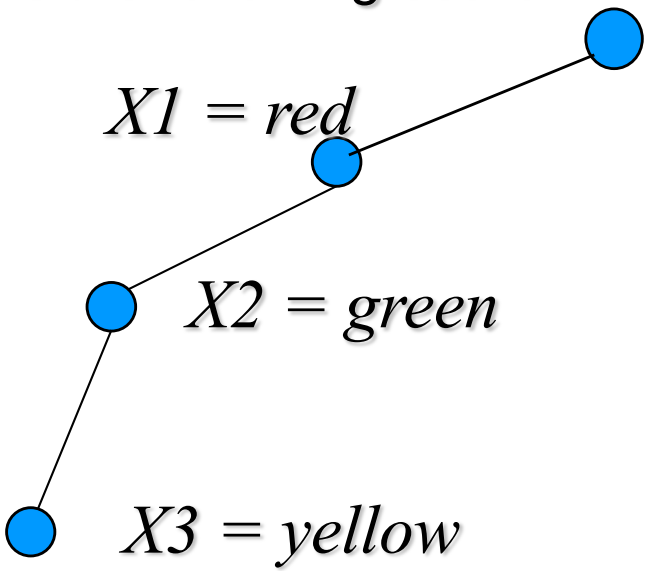
- PLA than FC does not increase the pruning of the tree in this case

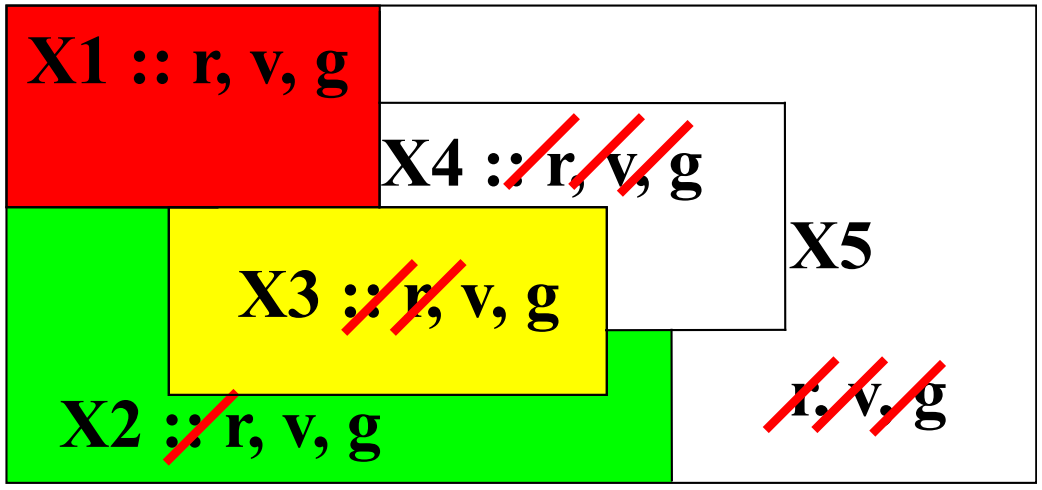




FC vs PLA for Map Coloring (*cont.*)

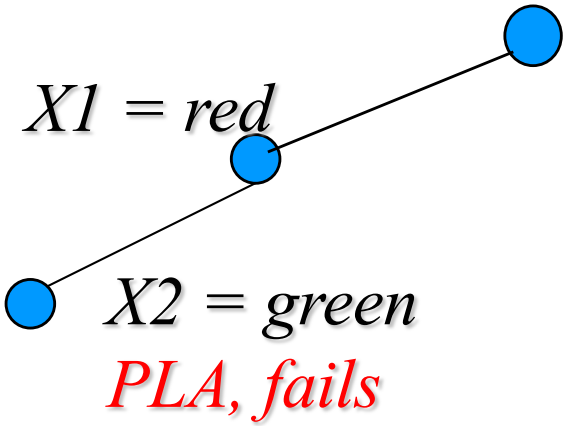
- Consider the case we have only three domain values (r, v, g), the FC performs the following search:





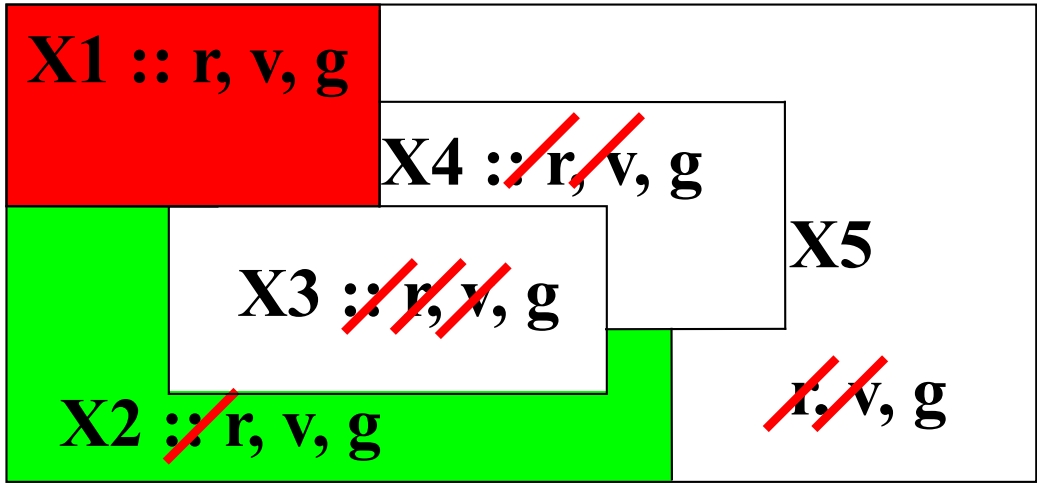
FC vs PLA for Map Coloring (*cont.*)

- The PLA anticipates the failure (of one level in this case):



The X3 domain becomes empty

There is no value in the domain of X4 compatible with the g value for X



LOOK AHEAD: EXAMPLE

- X0 < X1 < X2 < X3 with domains for X1, X2, X3 :: [1,2,3]

PLA, verification:

- for each value in D1 checks if there exists at least one value in D2 and at least one value in D3 compatible (in case these values do not exist we delete the value from the domain D1)
- for each value in D2 checks if there exists at least one value in D3 compatible (in case this value does not exist we delete the value from the domain D2)

FLA, beside the checks performed by PLA checks also:

- for each value in D2 checks if there is at least a value compatible D1 for each value in D2 checks if there exists at least one value in D3 compatible (in case this value does not exist we delete the value from the domain D2)
- for each value in D3 checks if there exists at least one value in D2 and at least one value in D1 compatible (in case these values do not exist, we delete the value from the domain of D3)

LOOK AHEAD: EXAMPLE (*cont.*)

- X0 <X1 <X2 <X3 domains with X1, X2, X3 :: [1,2,3]

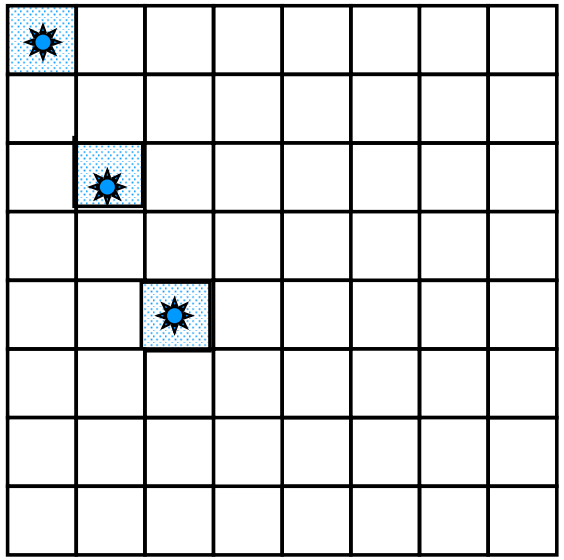
- PLA X1 :: [1,2]
  X2 :: [1,2]
  X3 :: [1,2,3]

- FLA X1 :: [1,2]
  X2 :: [2]
  X3 :: [3]

Note: one single iteration
We could do more!!

PLA: THE EIGHT QUEENS

- Suppose you have assigned to the first three X variables X_1, X_2, X_3 respectively the values 1,3,5.
- Domains remain the same of the FC:



- X4 is associated with D4 = (2,7,8)
- X5 is associated with D5 = (2,4,8),
- X6 is associated with D6 = (4),
- X7 is associated with D7 = (2,4,6),
- X8 is associated with D8 = (2,4,6,7).

PLA: THE EIGHT QUEENS

- X4 is associated with D4 = (2,7,8)
- X5 is associated with D5 = (2,4,8),
- X6 is associated with D6 = (4),
- X7 is associated with D7 = (2,4,6),
- X8 is associated with D8 = (2,4,6,7).

- We consider for D_4 the value 2. The values 4 in D_5, 6 in D_7 and 4 in D_8 are compatible with the value of 2 in D_4. However, the value 4 in D_6, is not compatible. Value 2 is deleted from D_4.

- For values 7 and 8 instead we find a support in the domain of other variables The domain associated with X_4 thus becomes: D_4= (7,8).

PLA: THE EIGHT QUEENS

- Likewise, we can delete value 4 from D_5 as it does not have support in D_6
- Value 8 in D_5 has support in the domain of future variables. So the domain associated with X_5 becomes:

  D_5= (2,8).

- The sets D_6 and D_7 are not modified.

- Note that we have removed the value 2 from the domain X_4 that was instead assigned by FC leading for a failure.

FLA: THE EIGHT QUEENS

- Take the earlier example, suppose you have already propagated domains with the PLA technique. Now we **apply FLA**:
  - X_4 is associated with the domain D_4= (7,8),
  - X_5 is associated with the domain D_5= (2,8),
  - X_6 is associated with the domain D_6= (4),
  - X_7 is associated with the domain D_7= (2,4,6),
  - X_8 is associated with the domain D_8= (2,4,6,7).
- Note that for the value 8 in D_5, there is no value belonging to D_4 that satisfies the constraints imposed by the problem.
- The PLA is not "aware" of this inconsistency because it verifies the consistency of the values belonging to a domain D_i with the values belonging to the domains Dj only if j> i.
- In the domain D_5, we have only the value 2. The value 4 in D_6 is compatible with 2 in D_5 and with the values 7 and 8 in D_4.
  - X_4 is associated with the domain D_4= (7,8),
  - X_5 is associated with the domain D_5= (2),
  - X_6 is associated with the domain D_6= (4),

FLA: THE EIGHT QUEENS

- Now consider the domain D_7: The value 2 in D_7 is not compatible with 2 in D_5 and is eliminated. The same thing happens for 4 in D_7 incompatible with 2 in D_5.
- D_7 is therefore: D_7= (6)
- The domain D_8, already at this point in the computation, does not contain more values compatible with those of the previous domains. In fact the value 2 in D_8 is incompatible with 2 in D_5, 4 in D_8 is incompatible with 4 in D_6 , 6 and 7 in D_8 are incompatible with 6 in D_7 . **The computation, therefore, fails without performing additional assignments**.
- The computational load due to additional verification of the consistency of constraints, needs to be balanced by a reduced search time. In the first levels of the tree sometimes FC is more effective.

TREE SEARCH: TO RECAP

- Consider a depth-first search. It assigns a variable at a time. At each step we either:
  - Find a solution
  - Discover a failure
  - Assign another variable

- The algorithm has three degrees of freedom:
  - the choice in the ordering of variables;
  - the choice in the ordering of values to be assigned to the current variable;
  - the propagation carried out in each node.

- The first two relate to search heuristics

TREE SEARCH: TO RECAP

- The third degree of freedom is what differentiates the different strategies:

- No propagation algorithms:
  - Generate and Test
  - Backtracking Standard

- Propagation Algorithms
  - Forward Checking
  - (Partial and Full) Look Ahead

SEARCH HEURISTICS

- While constraint propagation is performed by the solver, the order of variable selection and value selection are available to the programmer.
- The heuristics can then act on these two degrees of freedom to try to ensure the achievement of a good solution in a reasonable time for even the most complex problems.

The heuristics can be classified into:

- VARIABLE SELECTION HEURISTICS
  - determine what should be the next variable to instantiate. The two most commonly used heuristics are the **first-fail** (a.k.a **MRV**: Minimum remaing Values) that chooses the variable with the variable with the smallest domain, and **most-constrained principle** who chooses the variable appearing in the largest number of constraints. **Rationale: most difficult variables are assigned first**.

- VALUE SELECTION HEURISTICS
  - determine what value to assign to the selected variable. There are no general rules here byt the rationale is try first those values that are most likely to succeed (**least constraining principle**).

CLASSIFICATION OF HEURISTICS

A further classification is as follows:

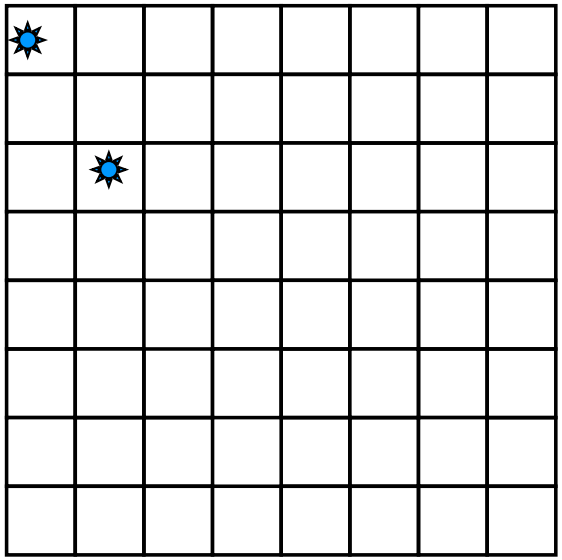
- Static heuristic:
  - determine variable and value order before starting the search; This order remains unchanged throughout the research.

- Dynamic heuristics:
  - select the next variable or value each time a new selection is done (so at every step of labeling).

- Dynamic heuristics are potentially better (less backtracking). The computation of the perfect heuristic (which requires no backtracking) is a problem that has, in general, the same complexity of the original problem. We must therefore find a tradeoff.

FORWARD CHECKING for 8 QUEENS (First Fail)

X_3 has the domain D_3= (5,6,7,8),
X_4 has the domain D_4= (2,6,7,8),
X_5 has the domain D_5= (2,4,7,8),
X_6 has the domain D_6= (2,4,5,8),
X_7 has the domain D_7= (2,4,5,6),
X_8 has the domain D_8= (2,4,5,6,7).



FORWARD CHECKING for 8 QUEENS

- The FC search assigns $X_3 = 5$ and propagates the constraints obtaining:

  X_4 has the domain D_4= (2,7,8),
  X_5 has the domain D_5= (2,4,8),
  X_6 has the domain D_6= (4),
  X_7 has the domain D_7= (2,4,6),
  X_8 has the domain D_8= (2,4,6,7).

- The domain of variable X_6 contains only one value and is chosen for the assignment

  X_4 has the domain D_4= (7,8),
  X_5 has the domain D_5= (2,8),
  X_7 has the domain D_7= (2,6),
  X_8 has the domain D_8= (7).

- The domain of variable X_8 contains only one value and is chosen for the assignment and so on.

CONSISTENCY TECHNIQUES

- In contrast to propagation algorithms that propagate the constraints as a result of instantiations of variables involved in the problem, consistency techniques reduce the original problem by eliminating domain values that cannot appear in a final solution.

- They can be applied statically or at every step of assignment (labeling) as powerful propagation techniques for the not yet instantiated variables.

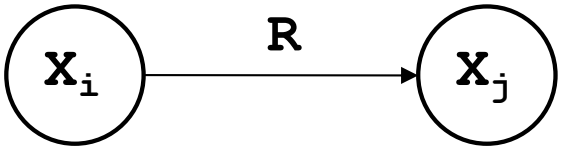
- All consistency techniques are based on a representation of the problem as a network (graph) of constraints. The arcs can be oriented or non-oriented: for example, the constraint > is represented by a directed arc, while the constraint ≠ from a simple arc (undirected or doubly oriented).

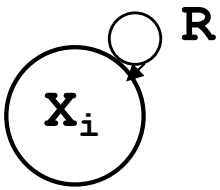
Basically the arc consistency is the iterative full look ahead

CONSTRAINT GRAPH

- For each CSP exists a graph (constraint graph) in which the nodes represent variables and the arcs are the constraints.
  - The binary constraints (R) connecting two nodes X_i and Xj:
  - The unary constraints are represented by arcs that begin and end on the same node X_i

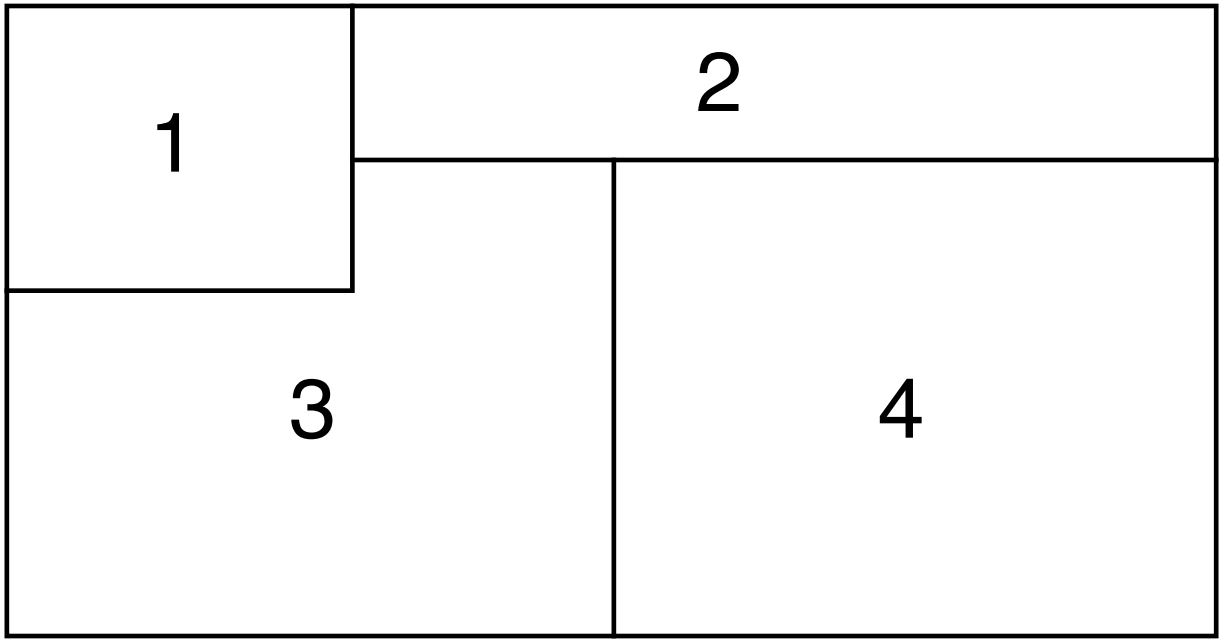
We will have only binary releations





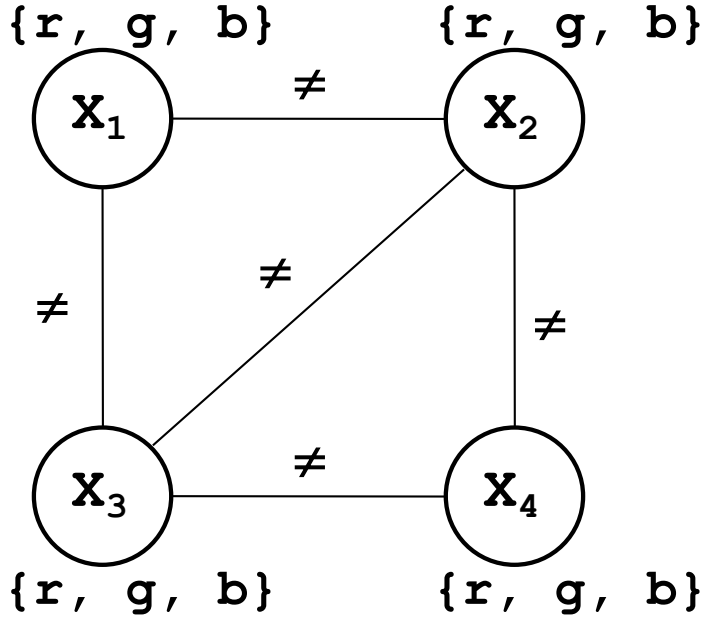
EXAMPLE: MAP COLORING PROBLEM

- Suppose we need to color of portions of a floor, characterized by a number, in such a way that two contiguous regions are colored by different colors. Suppose we also have available the colors red (R), green (g) and blue (b)

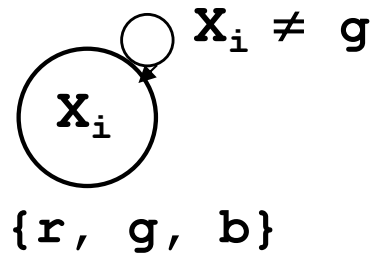


EXAMPLE: MAP COLORING PROBLEM

- The constraint-graph is as follows. However, there are combinations of values not compatible with each other (eg: X1=r, X2 = r, X3=r, X4 = r).
- There are several algorithms that implement different degrees of consistency.



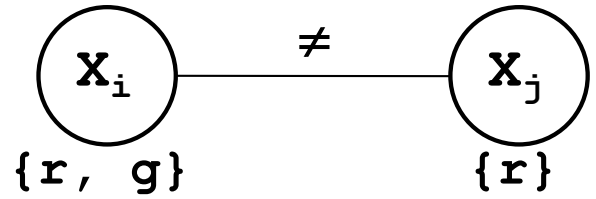
NODE CONSISTENCY



- NODE-CONSISTENCY: consistency level 1
  - A node of a graph of constraints is consistent if for each value $X_{the} \in D_{the}$ The unary constraint on X_{the} he is satisfied.

- In the above example, the node is not consistent because the value $g \in D_i$ violates the unary constraint on X_i.
- To make the node consistent it is necessary to eliminate from the domain of X_i the value g.
- A graph is node consistent if all its nodes are consistent.

ARC CONSISTENCY

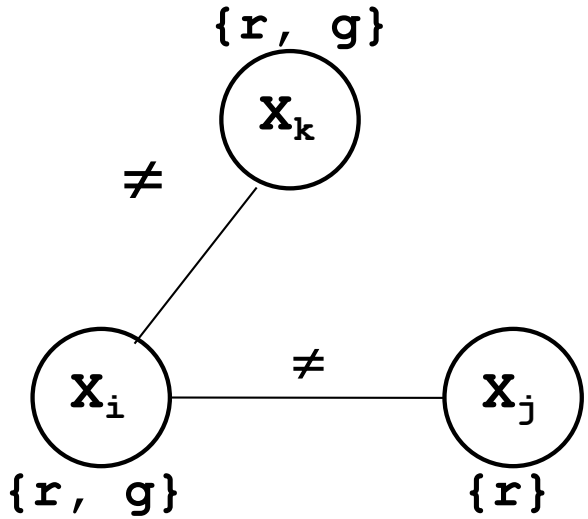


- The consistency of level 2 is obtained from a node-consistent graph and applies to arcs.

- ARC CONSISTENCY: an arc a(i, j) is consistent if for each value $x \in D_i$ It exists at least one value $y \in D_j$ such that the constraint between i and j is satisfied. y is called **support** for i

- The arc in the example above is not consistent because, considering the value $r \in D_i$, there is no value belonging to D_j which satisfies the constraint between them.
- To make consistent the arc between and X_j it is necessary to delete the value of r from the domain of X_i
- **This value would not appear in any feasible solution.**

ITERATIVE PROCEDURE

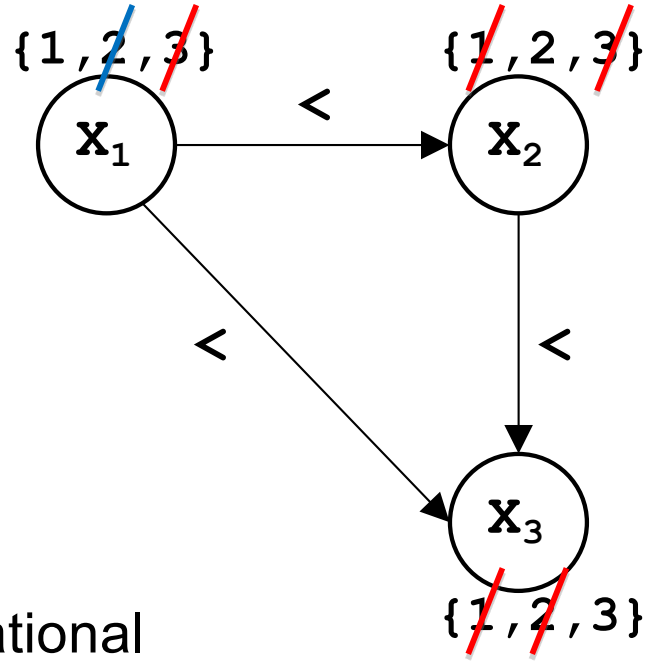


- A network is arc-consistent if every arc is arc consistent.
- The removal of a value from the domain of a variable makes further checks needed resulting in an iterative process.
- So this process should be repeated until the network reaches a stable configuration QUIESCENT
- Consider the arc between X_i and X_k this constraint is arc consistent
- Now the constraint between Xi and Xj removes r from the the domain of Xi which in turn requires the first constraint to be reconsidered
- Arc consistency is an iterative process that converges to a stable and arc-consistent network.

ARC CONSISTENCY: EXAMPLE (*cont.*)

- X1 <X2 <X3 domains with X1, X2, X3 :: [1,2,3]

- Before iteration:
- Second iteration:
- Third iteration: quiescence



The FLA led to domains:
X1 :: [1,2]
X2 :: [2]
X3 :: [3]
Less *pruning vs AC*, But lower computational cost (FLA is also called AC1 / 2)

ARC CONSISTENCY

- The arc consistency can be applied
  - before the search, as *preprocessing* to produce a *simplified problem with the same solutions of the original one* or
  - as propagation step (as done for Full *look ahead*) after each variable assignment it is often called *Maintaining Arc Consistency* (MAC)

- Many algorithms have been proposed. The most famous is called AC-3 (Mackworth, 1977)

- Use a queue of arcs (*queue*), and cycles until it is empty

- As soon as the domain of a variable Xi is reduced, we add in *queue* all arcs (Xk, Xi) for each variable Xk connected by an arc with Xi.

AC-3 algorithm (Mackworth, 1977)

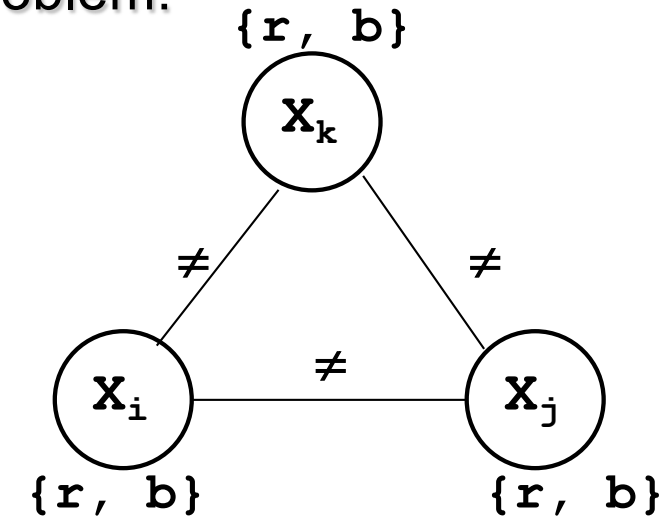
```
function AC-3( csp) returns the CSP, possibly with reduced domains
  inputs: csp, a binary CSP with variables {X_1, X_2, ..., X_n}
  local variables: queue, a queue of arcs, initially all the arcs in csp

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

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

OBSERVATION

- Suppose we consider the network of constraints related to an instance of map coloring problem:



- This network is arc-consistent: in fact, for each value of each domain, there is at least a value in every other domain that matches the existing constraint between the two nodes.
- However, it is immediate to see that the network has no solution
- **All values removed by arc-consistency are not part of any feasible solution BUT values that are left in the domains are not necessarily part of a consistent solution**

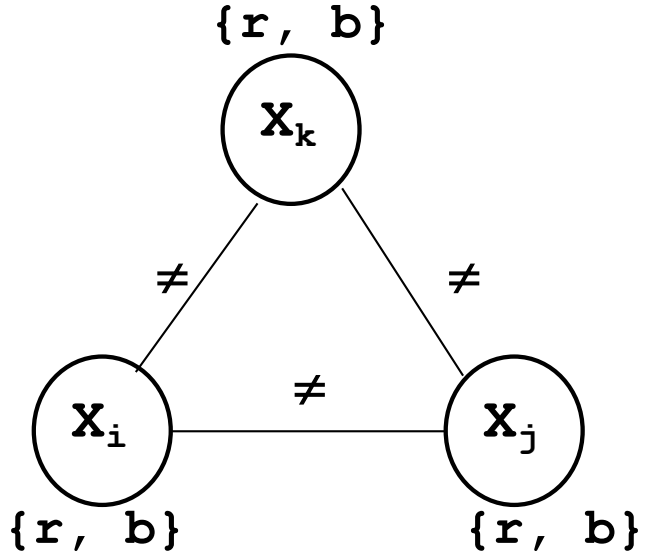
PATH-CONSISTENCY: level 3

- The consistency of level 3 is obtained starting from an arc-consistent graph.

- PATH CONSISTENCY: A path between the nodes (i, j, k) is path consistent if, for every value $x \in D_i$, and $y \in D_j$ (that are node and arc-consistent) there is a value $z \in D_k$ that satisfies the constraints P (i, k) **and** P (k, j).

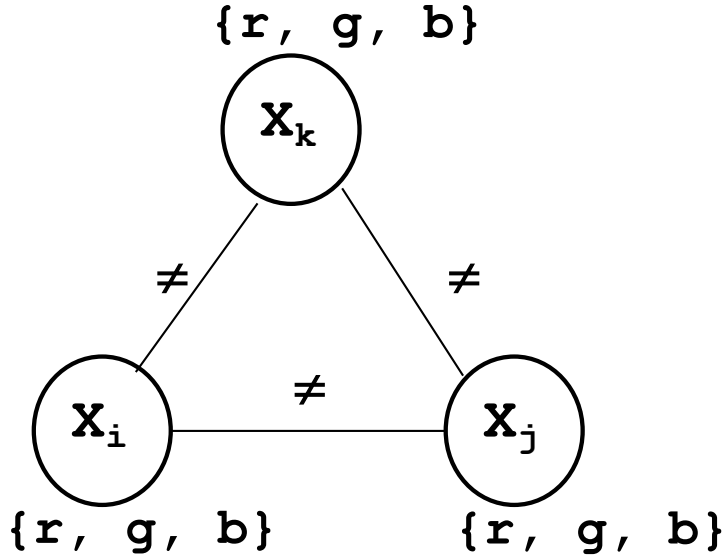
(The consistency of the unary constraint P (k) is guaranteed by the node consistency)

PATH CONSISTENCY: EXAMPLE 1 (*cont.*)



- Additional data structures needed, for each pair of variables (for each arc) we need to store pairs of compatible values for which exist a support in the domain of any third variable (example: for the arc Xi to Xj, none of the pairs <r, b> and <b, r> has a support value in the domain Xk)
- The network cannot be made path consistent thus leading to a failure.

PATH CONSISTENCY: EXAMPLE 1 (*cont.*)



- By adding a color to the available ones: r, g, b
- The network is PC
- Check the consistency of level n = 3 for this network that has exactly n = 3 variables is equivalent to verify that *there is a solution AND all values that are left in the domain are part of a consistent solution*
- This happens because the level of consistency is equal to the number of variables

K-CONSISTENCY

- In principle, for a CSP of k variables, if we want to apply consistency techniques and obtain in variable domains only those values that are part of a feasible solution, we need to apply k-consistency
- For each k-1-tuple of values consistent with the constraints imposed by the problem, there is a value for each k-th variable that satisfies the constraints among all k variables.
- In general, if a graph containing n variables is k-consistent with k <n, then search in the remaining space is needed.
- If a CSP containing n variables is n-consistent, then you can find a solution without search.
- Freuder in 1978 has defined a general algorithm to make a network k-consistent for any k.
- However, making n-consistent a network of n variables has an complexity exponential in n (*the cost is the same as solving the original problem*)

CONSTRAINT SOLVERS in practice

- Constraint solvers are tools to solve CSPs that embed all techniques seen so far. Search and proapagation. They typically use *arc-consistency* for the propagation of constraints and search.
- They are based on the so called Constraint Programming languages

- The constraints are seen as software components that encapsulate an *filtering* algorithm. Very often the *filtering* algorithm making the propagation is not *general purpose* like those seen so far, but is based on the semantics of the constraint for efficiency reasons.

- Example: X :: [1..10], Y :: [1..10], X> Y we don't have to check all values in the two domains, but we just have to check out *bound*. In particular, this constraint is AC if min (X)> min (Y) and max (X)> max (Y)

- Example X :: [1..10], Y :: [1..10], X $\neq$Y. This constraint is ALWAYS AC if the two domains contain more than one value. Therefore the constraint is propagated only when one of the two variables is instantiated to a value. This value is removed from the domain of the other.

CONSTRAINT SOLVER in practice

- A key feature of the constraint solver is the presence of n-ary constraints, also called GLOBAL constraints
- Also they embed a filtering algorithm
- Clearly to achieve the consistency of a n-ary constraint (Generalized Arc Consistency), in principle, you should apply the n-consistency, which has exponential complexity.
  - However, there are particular constraints for which reaching the n-consistency has polynomial cost (eg **AllDifferent)**
  - For others an approximation of Generalized Arc Consistency is achieved

CSPs and OPTIMIZATION

- We considered only CSPs in which variables have discrete domains. Most of these problems are NP-hard, that are problems for which has not yet been found, and probably does not exist, an algorithm able to find the solution in polynomial time in the size of the problem.

- A Constraint Optimization Problem (COP) is a constrained problem in which an objective function is added. A COP is then formally described as a CSP whose purpose is not only to find a feasible solution, but the optimal solution according to a certain evaluation criterion.

CSPs and OPTIMIZATION

- The general algorithm to solve a CSP can then be used to solve any COP. In fact, after having described the problem in terms of variables, constraints and domains, just add a further variable that represents the objective function.

- Whenever a CSP solution is found, a new constraint is added that ensures that any future solution has a better value of the objective function. This process continues until you can no longer find any solution.

- The last solution found is the optimal solution.