Constraint Programming Search

Eric MONFROY

IRIN, Université de Nantes

Objectives

- recap : CSPs and solving CSPs
- notion of search trees
- discuss various search mechanism (for enumeration)
  - backtrack
  - forward checking
  - partial look ahead (maintenaing arc consistency –MAC)
  - full look ahead (maintenaing arc consistency –MAC)
  - discuss search mechanism for constrained optimization
  - discuss search heuristics

Search trees

CSP (recap)

A *constraint satisfaction problem* (CSP) is defined by :

- a sequence of variables $X = x_1, \ldots, x_n$ with *domains* $D_1, \ldots, D_n$ (associated to the variables)
- a set of constraints $C_1, \ldots, C_l$, each C_i on a sub-sequence Y_i of X

A *solution of the CSP* is a n-tuple d such that :

- $d \in D_1 \times \cdots \times D_n$
- and for each i, $d[Y_i] \in C_i$

Constraint propagation (recap)

- replace a CSP by a CSP which is :
  - equivalent (same set of solutions)
  - “smaller” (domains are reduced)
  - “simpler” (constraints are reduced)
- constraint propagation mechanism :

  repeatedly reduce domains or constraints
- incomplete solver

Constraint solving framework (recap)

```
solve(CSP) :
        while not finished do
            pre-process
            constraint propagation
            if happy
               then finished=true
               else split
                     part-of search
            endif
        endwhile
```

where part-of search consists in calls to the solve function

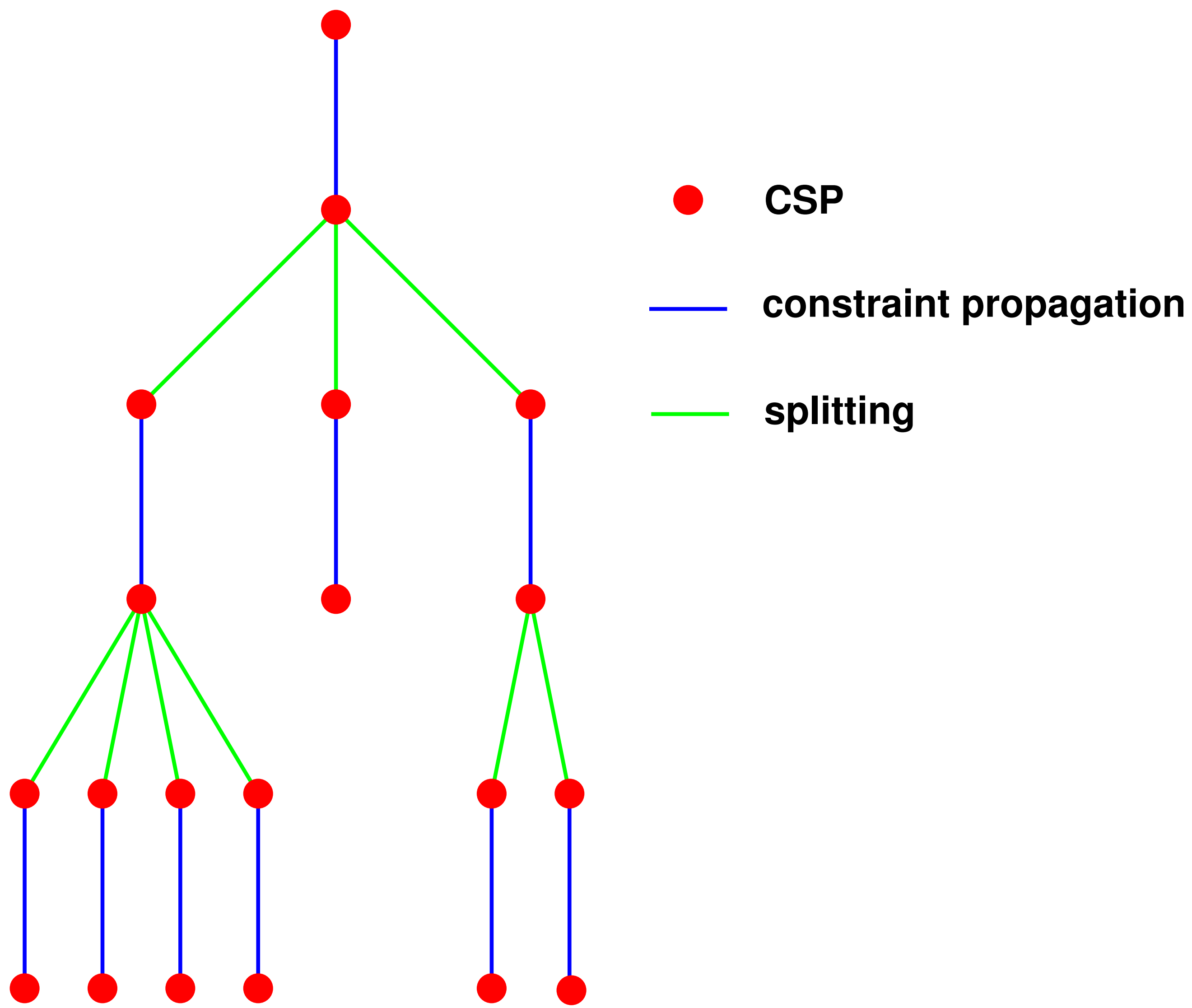
Remark : part-of search is one of the mechanisms defining search

Search trees (1)

the solving process can be seen as a *search tree* s.t. :

- nodes are CSPs
- the root is the initial CSP
- an arc is either :
  - a constraint propagation phase
  - or split phase

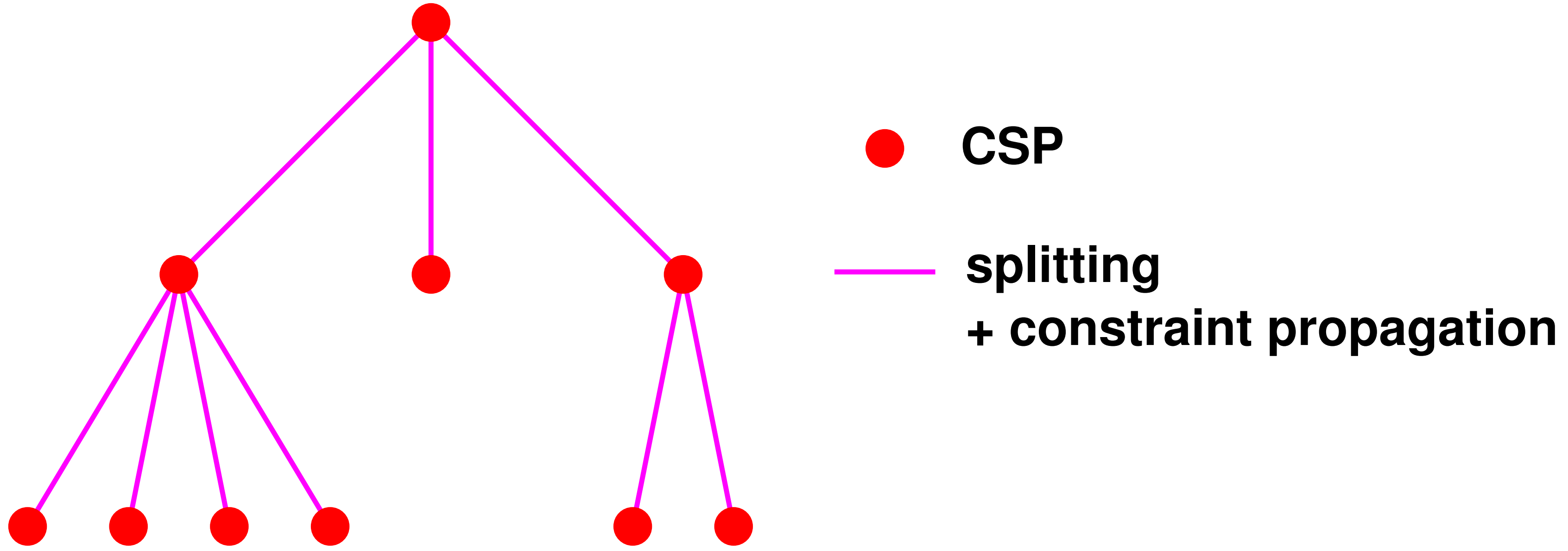
Search trees (2)



Search trees (3)

constraint propagation and splitting are often joined to reduce trees

an arc is a split followed by constraint propagation



Search trees (4)

- several search trees to solve a CSP, depending on
  - constraint propagation
  - search-part
  - splitting :
    - ordering of variables
    - type of splitting (enumeration, bisection, . . .)
    - value selected (in case of enumeration)
- solutions with different search trees :
  - same solutions when look for all
  - can be different when look for ONE solution
- efficiency and memory : **huge differences**

Search algorithms

n-Queens (recap)

Place n queens on a $n \times n$ board so that they do not attack each other

Modelling :

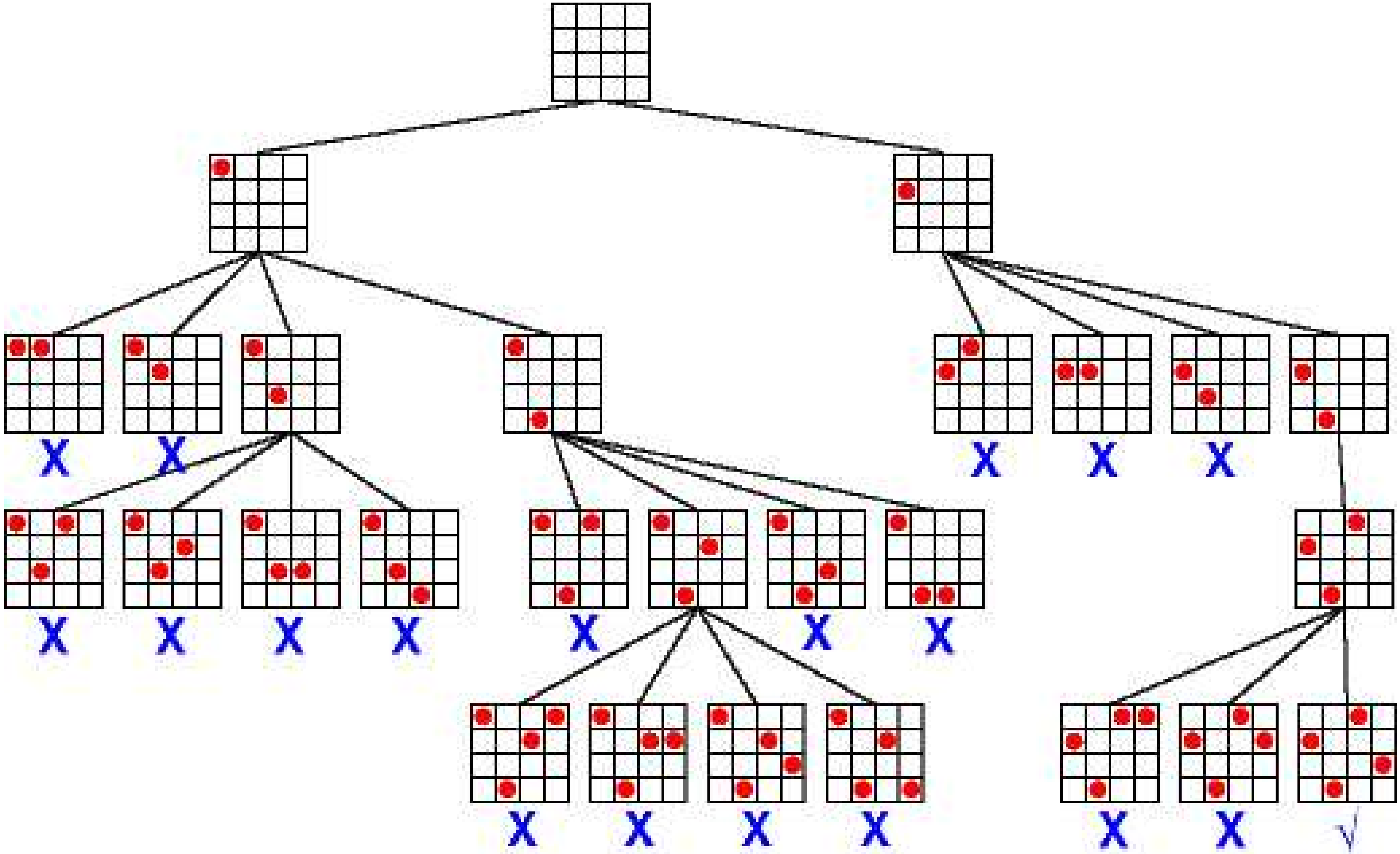
- **Variables** : $c_1, \ldots, c_n$

  one per column : the value of c_i represents the line where the queen is in the column
- **Domains** : [1..n]
- **Constraints** : for $i \in [1..n-1)$ and $j \in i+1..n]$
  - not two queens on the same line : $x_i \neq x_j$
  - not 2 queens on the same SW-NE diagonal : $x_i \neq x_j + j - i$
  - not 2 queens on the same NW-SE diagonal : $x_i \neq x_j + i - j$

Backtracking (a la Prolog)

- no constraint propagation phase
- full enumeration during a splitting phase (since no propagation)
- generally : depth-first, left-first search (i.e., exploration of the search tree)

4-queens by backtracking



schema issued of *Online guide to Constraint Programming. R. Barták*

Forward checking

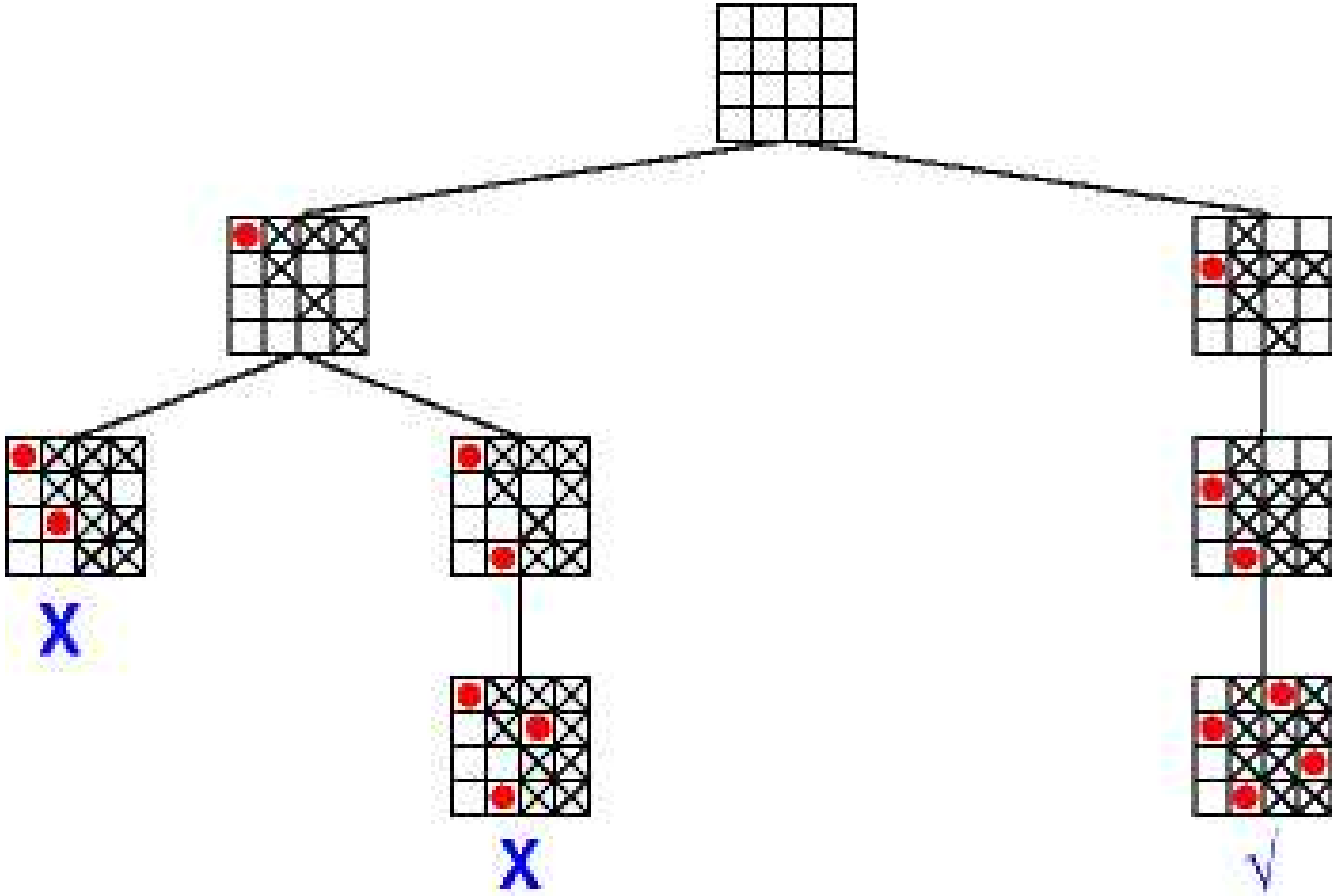
Forward checking

- split : enumeration
- constraint propagation :
  - first time :

    generally, complete propagation (AC-like algorithm)
  - in the loop :

    after each instanciation of a variable x : remove from each variable y (not yet instanciated) values inconsistent w.r.t. constraints containing x and y

4-queens by forward checking



schema issued of *Online guide to Constraint Programming. R. Barták*

Partial look ahead

Partial look ahead

- split : enumeration
- constraint propagation :
  - first time :

    generally, directed or complete propagation (directional or AC-like algorithm)
  - in the loop :

    directional arc-consistency

    i.e., propagation directed by variable ordering

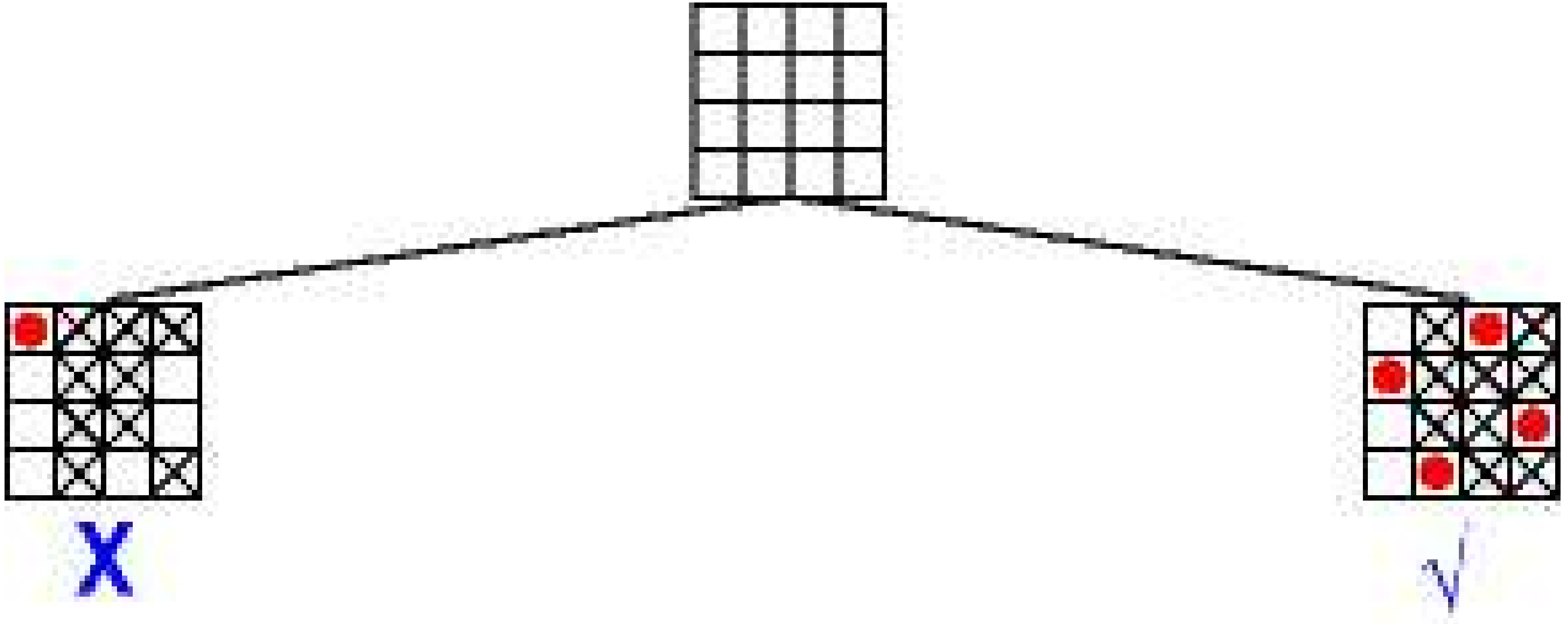
    (no fixed point)

Full look ahead

Full look ahead
(or Maintaining Arc consistency = MAC)

- split : enumeration
- constraint propagation :
  - first time :
    generally, complete propagation (AC-like algorithm)
  - in the loop :
    arc-consistency (or hyper-arc consistency)
    (fixed point of reductions)

4-queens by full look ahead



schema issued of *Online guide to Constraint Programming. R. Barták*

Constrained optimization

Constrained optimization problems

Optimisation = minimization or maximization

find

$$
\begin{array}{rll}
 & \max f(x_1, \ldots, x_n) & \textit{maximization problem} \\
\textit{or} & \min f(x_1, \ldots, x_n) & \textit{minimization problem}
\end{array}
$$

under the constraints

$$
\begin{cases}
c_1(x_1^1, \ldots, x_{k_1}^1) \\
\ldots \\
c_m(x_1^m, \ldots, x_{k_m}^m)
\end{cases}
$$

Example : knapsack problem (1)

a smuggler has a 9 unit capacity knapsack. He wants to smuggle : whisky, perfume, and cigarette packs. We have :

Product	volume (in units)	profit
whisky	4	15 €
perfume	3	10 €
cigarettes	2	7 €

a travel is interesting if the smuggler gains at least 30 € .

What should he carry ?

Knapsack problem (2)

Modelling :

- let W, P, C be the number of bottles of whisky, parfum and packs of cigarettes
- constraint on capacity : $4W + 3P + 2C \leqslant 9$
- constraint on profit : $15W + 10P + 7C \geqslant 30$

Knapsack problem (3)

program :

```
goal(W,P,C):-
   [W,P,C]::[0..9],
   4*W + 3*P + 2*C #=< 9,
   15*W + 10*P + 7*C #>= 30,
   labeling([W,P,C]).
```

answers :

- bound consistency : $W \in [0,2], P \in [0,3], C \in [0,4]$
- enumeration : $(W,P,C) = (0,1,3)$, $(W,P,C) = (0,3,0)$, $(W,P,C) = (1,1,1)$, $(W,P,C) = (2,0,0)$

Knapsack problem (4)

Solution maximizing the profit ?

```
goal(W,P,C):-
   [W,P,C]::[0..9],
   4*W + 3*P + 2*C #=< 9,
   15*W + 10*P + 7*C #>= 30,
   labeling([W,P,C]).

maxgoal:-
    Profit #= 15*W + 10*P + 7*C,
    Loss #= -Profit,
    minimize(goal(W,P,C),Loss),
    write([W,P,C,Profit]).
```

Solution : `Profit = 32`, with `(W,P,C)=(1,1,1)`

Maximization

branch and bound procedure : to maximize Profit

- search for a first solution : Pr_1
- add the constraint Profit $> Pr_1$
- update current bound and best bound
- *backtrack*
- at the end, re-computation with the best bound

adding the constraint Profit $> Pr_1$

➔ pruning solutions with worse profit

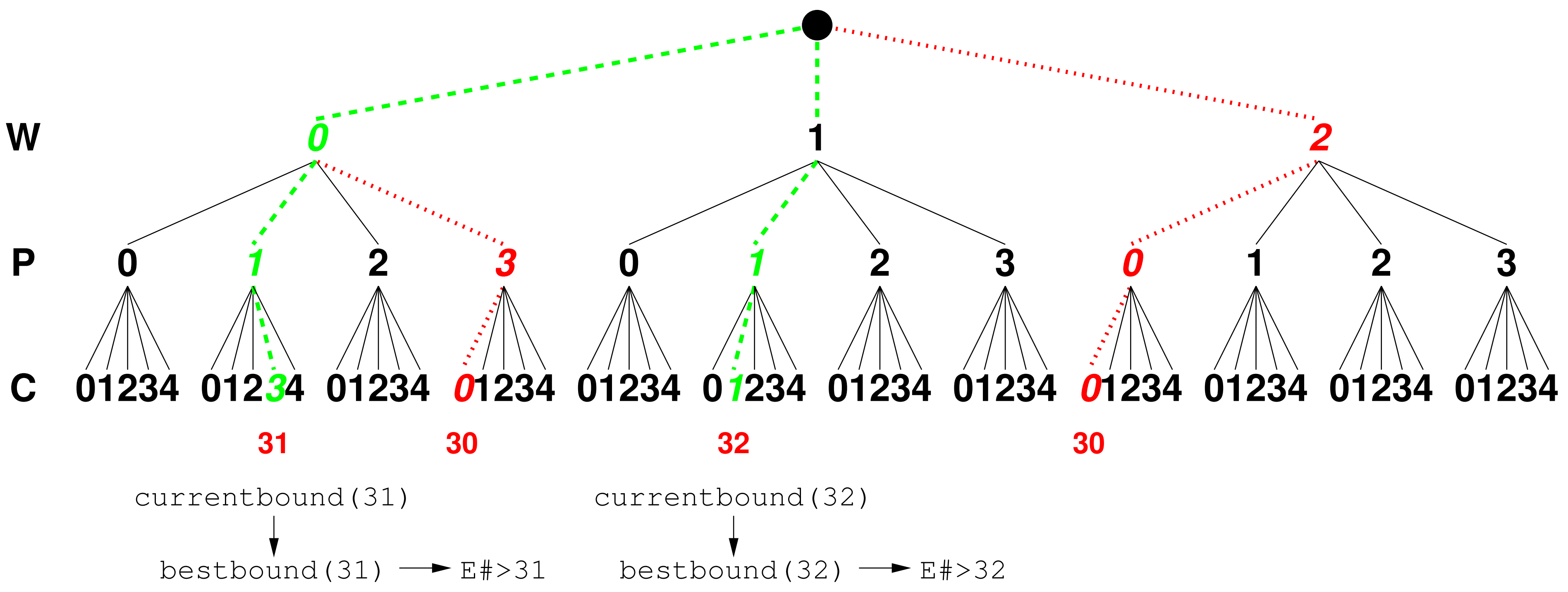
Back on the smuggler (1)

detailed solution in ECL^{i}PS^{e} :

```
[eclipse 30]: maxgoal.
Found a solution with cost -31
Found a solution with cost -32
[1, 1, 1, 32]
Yes (0.00s cpu)
```

Back on the smuggler (2)

before enumeration : $W \in [0,2], P \in [0,3], C \in [0,4]$



maximize/2

```
maximize(G,E):-
        get_min_value(G,E,M),
        E #= M,
        call(G).

get_min_value(G,E,_):-
        apply_new_bound(E),
        once(G),
        record_better_bound(E),
        fail.

get_min_value(_,_,M):-
        retract(bestbound(M)).

```

```
apply_new_bound(_).
apply_new_bound(E):-
        retract(currentbound(B)),
        asserta(bestbound(B)),
        E #> B,
        apply_new_bound(E).

record_better_bound(E):-
        (retract(bestbound(_))
          -> true ; true),
        asserta(currentbound(E)).
```

save the best solution and the current solution as facts

the goal `G` **must** instanciate `E` !

Search heuristics

Search heuristics

- not relevant for solutions
  (except when looking for ONE solution)
- crucial for efficiency
- heuristics at several levels :
  - variable to split
  - splitting mechanism (bisection, enumeration, . . .)
  - where to split (bisection)
  - values of variables (for enumeration)
- combination of heuristics
  (e.g., mix selection with several criteria)

Variable selection

- select the variable with the smallest domain (fail first)
- select the variable with the largest domain (reduce first)
- select the most constrained variable
  - most important variable in the problem
  - biggest number of possible reductions (most constrained)

Value selection (enumeration)

- select the smallest value
- select the largest value
- select the middle value