

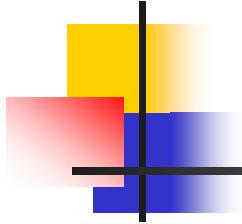
# Extensions de méthodes à divergences limitées pour la résolution de problèmes de décision

W. Karoui <sup>(1,2)</sup>, M.J. Huguet <sup>(1)</sup>, P. Lopez <sup>(1)</sup>, W. Naanaa <sup>(3)</sup>

<sup>(1)</sup> LAAS-CNRS, Université de Toulouse, France

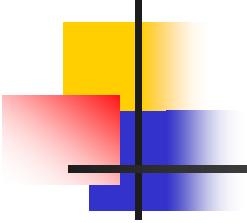
<sup>(2)</sup> Unité de recherche ROI, Ecole Polytechnique de Tunisie

<sup>(3)</sup> Faculté des Sciences de Monastir, Tunisie



# Introduction and motivations

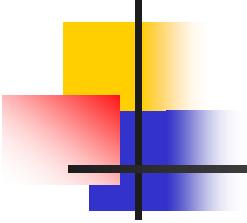
- Binary CSPs with finite discrete domains
  - First solution
- Proposed approach: YIELDS, NG-YIELDS
  - Tree search methods based on:
    - Discrepancies (LDS family)
    - Propagation
  - Purpose: To reduce discrepancies number



# Outline

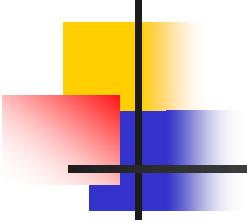
---

- **Background**
- **Proposed approaches**
- **Experimental results**
- **Related works**
- **Conclusion and further works**



# CSP search methods

- CB-FC and CB-AC (MAC)
- Limited Discrepancy Search (LDS)
  - 1995 – Harvey & Ginsberg
  - Instantiation heuristics
    - Variable ordering
    - Value ordering
  - Discrepancy
  - Instantiation failure
    - Leaf  $\neq$  solution
    - Diverge / Value ordering
  - Search space exploration in the increasing order of discrepancies



# Limited Discrepancy Search

- LDS: General principle

```
Nb_Div ← 0;
```

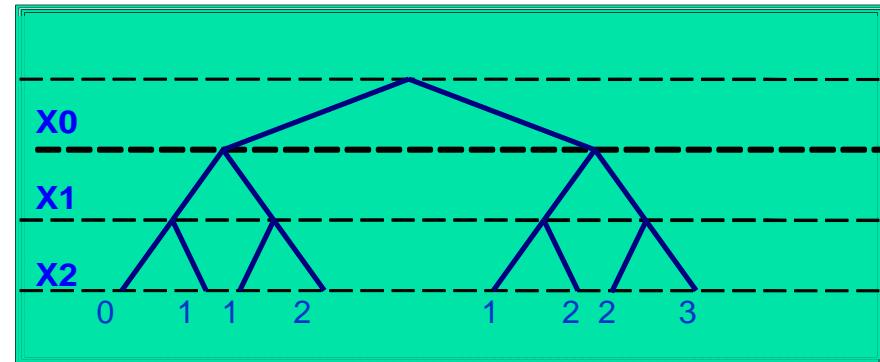
```
While Nb_Div < Nb_Max and no solution Do
```

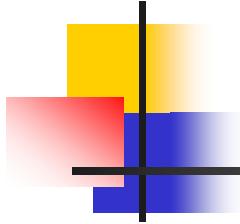
```
    Sol ← Instantiate ( (X, D, C), Nb_Div)
```

```
    If Sol = Null then
```

```
        Nb_Div ← Nb_Div + 1
```

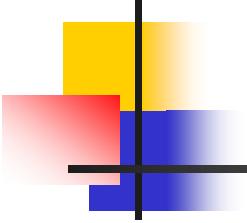
- Stop: first solution
- Example:  
(binary variables)





# Limited Discrepancy Search

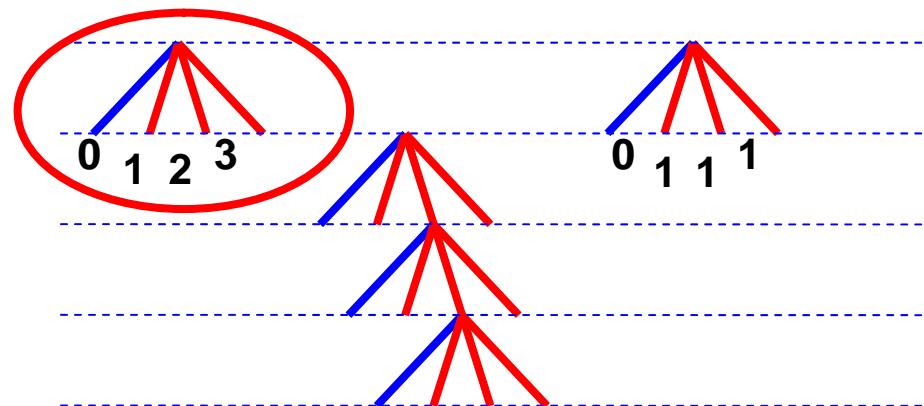
- LDS Characteristics:
  - + To avoid “early mistakes”
  - Too redundant search trees
- ➔ Improvements: ILDS [Korf 1996]  
DDS [Walsh 1997]

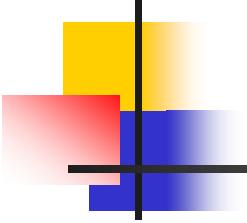


# Limited Discrepancy Search

- LDS for CSPs with non-binary variables

How to count  
discrepancies  
?





# YIELDS

---

- Yet ImprovEd Limited Discrepancy Search
- YIELDS: General principle

Nb\_Div  $\leftarrow$  0;

While Nb\_Div < Nb\_Max and no solution Do

Sol  $\leftarrow$  Instantiate ( (X, D, C), Nb\_Div, Weight)

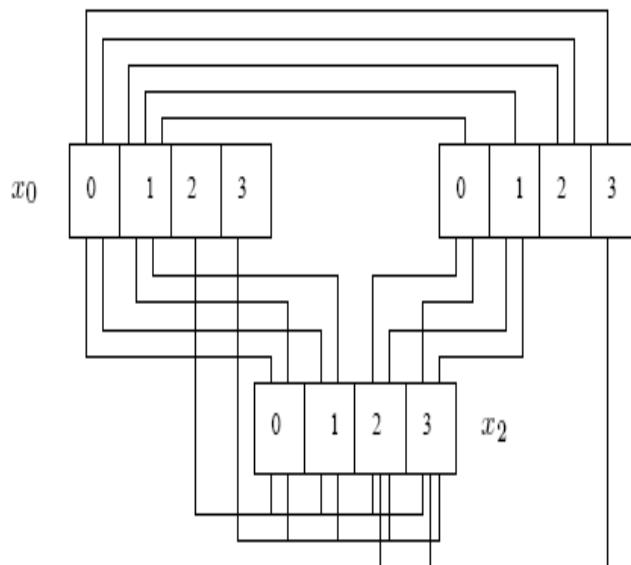
If Sol = Null then

Nb\_Div  $\leftarrow$  Nb\_Div + 1

Update (Weights)

- Stop: first solution

# Example 1



Incompatibility diagram

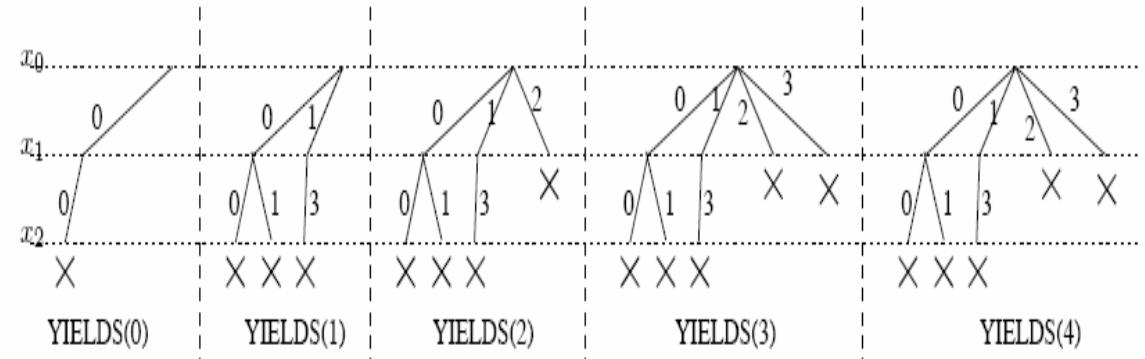


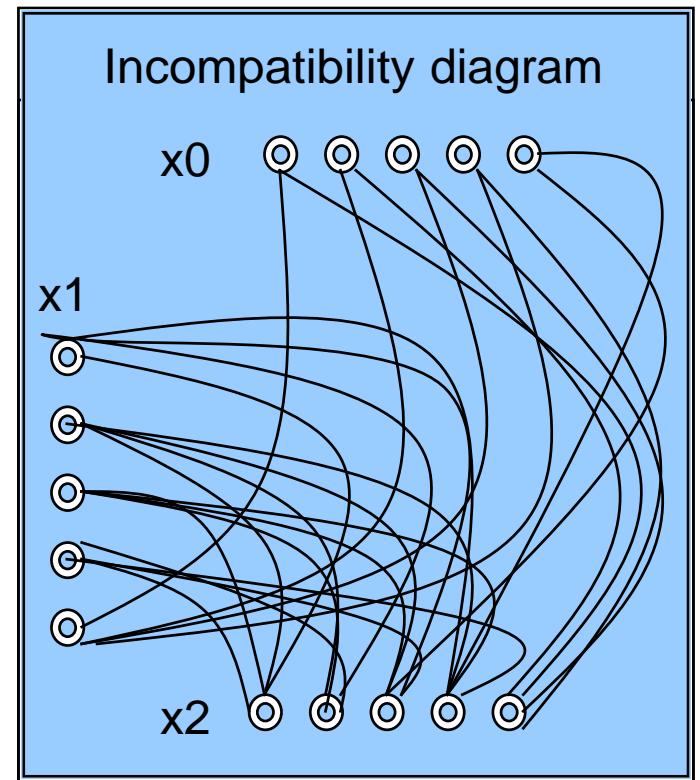
Illustration of **YIELDS**

Weight	Initial	<b>YIELDS(0)</b>	<b>YIELDS(1)</b>	<b>YIELDS(2)</b>	<b>YIELDS(3)</b>	<b>YIELDS(4)</b>
$W[x_0]$	0	1	2	3	4	4
$W[x_1]$	0	1	3	3	3	3
$W[x_2]$	0	0	0	0	0	0

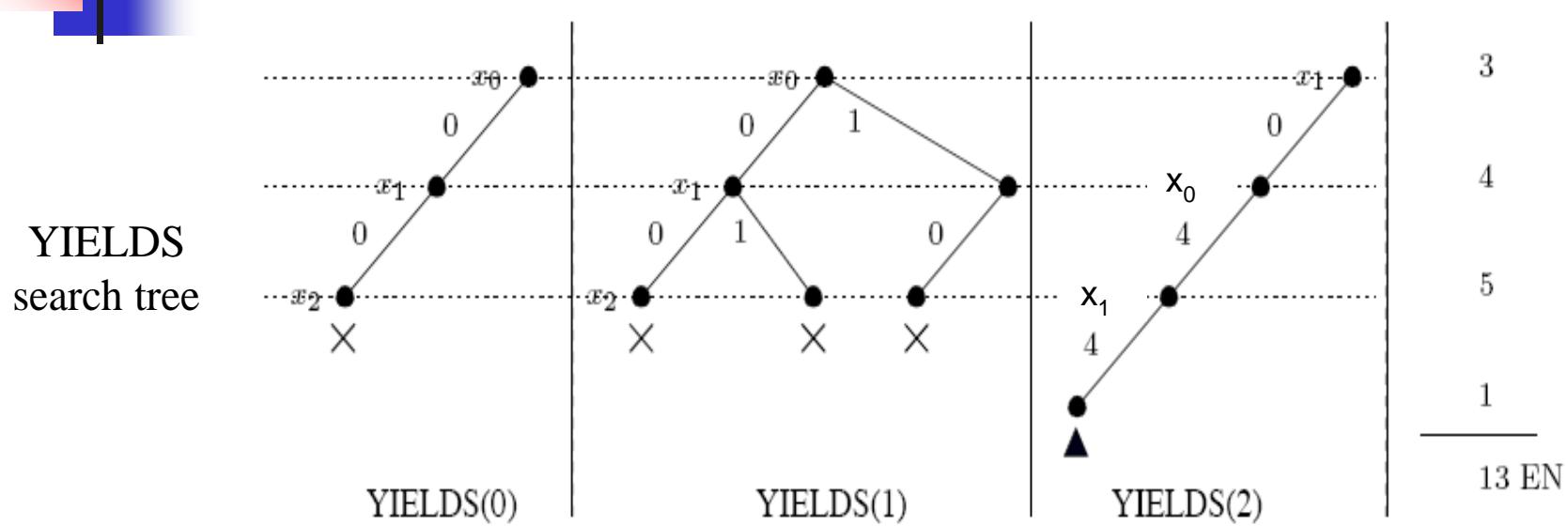
Variable weights

# Example 2

- Let consider a CSP defined by:
  - $X = \{x_0, x_1, x_2\}$
  - $D = D = \{D_0, D_1, D_2\}$  avec  $D_0=D_1=D_2=\{0, 1, 2, 3, 4\}$
  - $C = \{(x_0, 0), (x_1, 4)\}, \{(x_0, 0), (x_2, 4)\}, \{(x_0, 1), (x_1, 4)\}, \{(x_0, 1), (x_2, 4)\}, \{(x_0, 2), (x_1, 4)\}, \{(x_0, 2), (x_2, 4)\}, \{(x_0, 3), (x_1, 4)\}, \{(x_0, 3), (x_2, 4)\}, \{(x_0, 4), (x_2, 2)\}, \{(x_0, 4), (x_2, 3)\}, \{(x_1, 0), (x_2, 0)\}, \{(x_1, 0), (x_2, 1)\}, \{(x_1, 0), (x_2, 2)\}, \{(x_1, 0), (x_2, 3)\}, \{(x_1, 1), (x_2, 0)\}, \{(x_1, 1), (x_2, 1)\}, \{(x_1, 1), (x_2, 2)\}, \{(x_1, 1), (x_2, 3)\}, \{(x_1, 2), (x_2, 0)\}, \{(x_1, 2), (x_2, 1)\}, \{(x_1, 2), (x_2, 2)\}, \{(x_1, 2), (x_2, 3)\}, \{(x_1, 3), (x_2, 0)\}, \{(x_1, 3), (x_2, 1)\}, \{(x_1, 3), (x_2, 2)\}, \{(x_1, 3), (x_2, 3)\}$



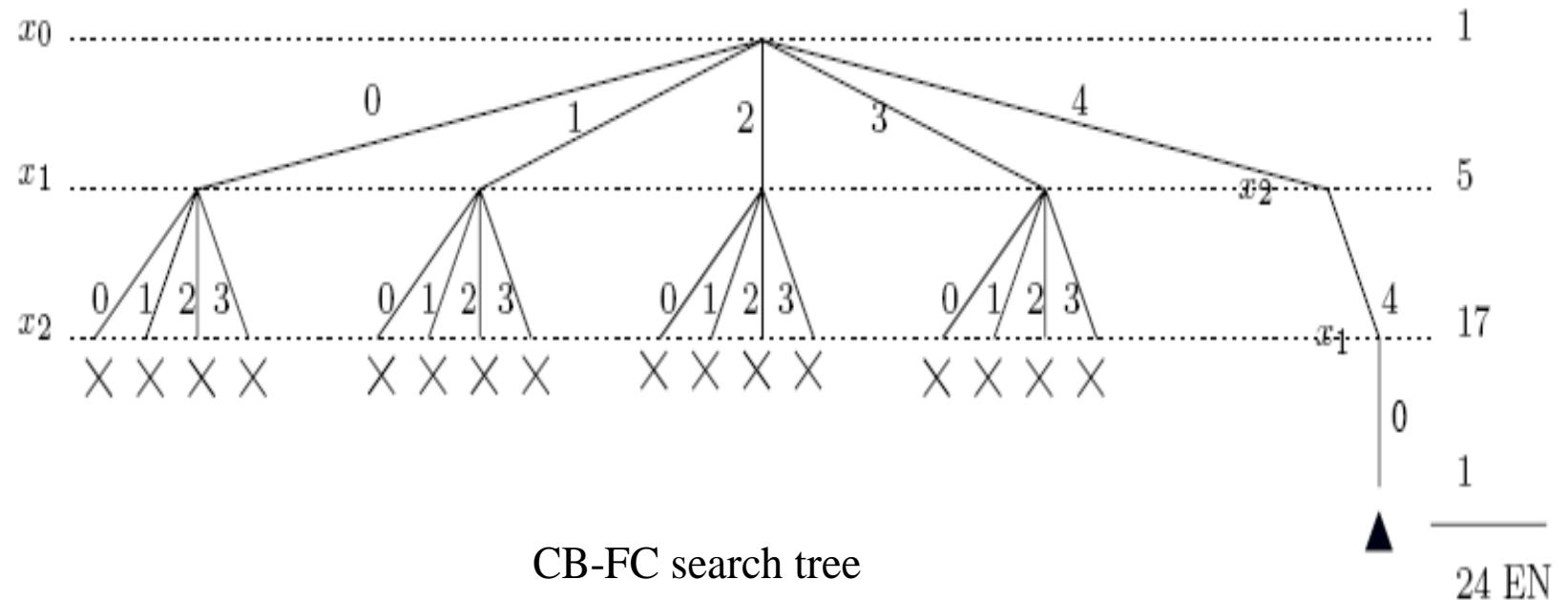
# Example 2

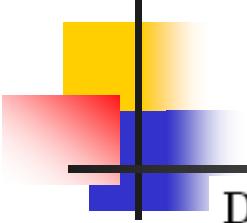


Weight	Initial	YIELDS(0)	YIELDS(1)	YIELDS(2)
$W[x_0]$	0	1	2	2
$W[x_1]$	0	1	3	3
$W[x_2]$	0	0	0	0

Weights

# Example 2





# Formalization

**Definition 1.** Let  $P = (X, D, C)$  be a binary CSP of  $n$  variables and  $w_i$  a weight associated to each variable  $x_i$ . The weight vector  $W$  of  $P$  is the vector composed of weights of all variables of the problem:

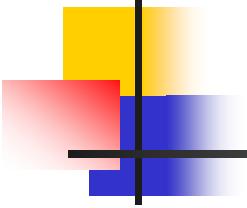
$$W(P) = [w_0, w_1, \dots, w_{n-1}]$$

**Definition 2.** Let  $W1$  and  $W2$  be two weight vectors of  $P$ , a binary CSP. The variation of weights of  $P$  is given by  $\Delta W$  the vector difference between  $W1$  and  $W2$ :

$$\Delta W(P) = W2(P) - W1(P)$$

**Proposition 1.** Let assume that variable weights are initially equal and that they are incremented every time that we do not found a variable value which respects the limit on the number of authorized discrepancies. Let consider two successive iterations of YIELDS for the resolution of  $P$  a binary CSP. If the variation of weights  $\Delta W(P)$  between these iterations is equal to the null vector, then we can be sure that:

1. The process of learning comes to end.
2.  $P$  is an intractable problem.



# YIELDS Algorithm

---

**Algorithm 4** YIELDS\_iteration( $X, D, C, k, Sol$ )

---

```
1: if  $X = \emptyset$  then
2:   return  $Sol$ 
3: else
4:    $x_i \leftarrow$  First_VariableOrdering( $X, Weight$ )
5:    $v_i \leftarrow$  First_ValueOrdering( $D_i, k$ )
6:   if  $v_i \neq NIL$  then
7:      $D' \leftarrow$  Update( $X \setminus \{x_i\}, D, C, (x_i, v_i)$ )
8:      $I \leftarrow$  YIELDS_iteration( $X \setminus \{x_i\}, D', C, k, Sol \cup \{(x_i, v_i)\}$ )
9:     if  $I \neq NIL$  then
10:      return  $I$ 
11:    else
12:      if  $k > 0$  then
13:         $D_i \leftarrow D_i \setminus \{v_i\}$ 
14:        return YIELDS_iteration( $X, D, C, k-1, Sol$ )
15:      else
16:         $Weight[x_i] \leftarrow Weight[x_i]+1$ 
17:        Exceed  $\leftarrow$  True // impossible to diverge
18:      end if
19:    end if
20:  end if
21:  return NIL
22: end if
```

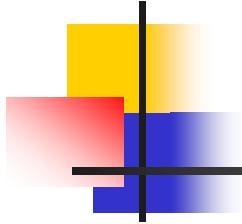
---

**Algorithm 3** YIELDS( $X, D, C, k_{max}, Sol$ )

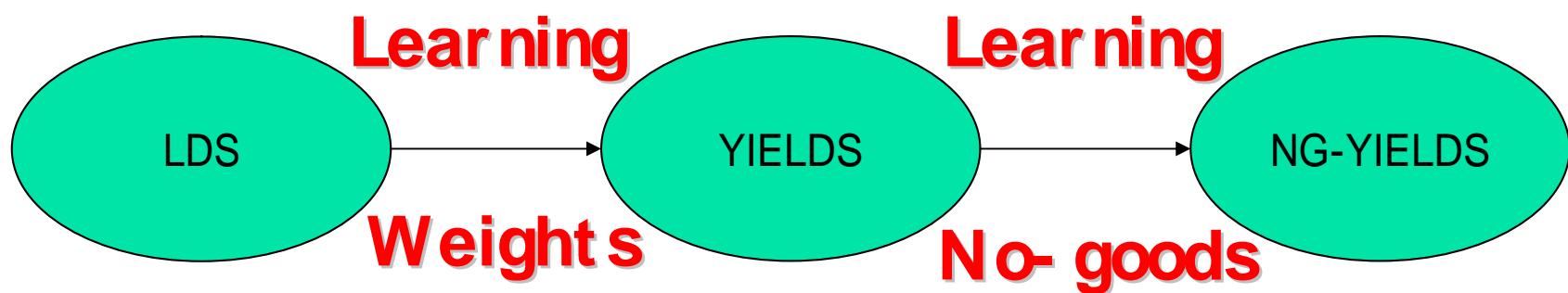
---

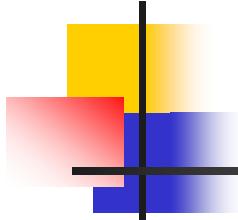
```
1:  $k \leftarrow 0$ 
2:  $Sol \leftarrow NIL$ 
3: Exceed  $\leftarrow False$ 
4: while ( $Sol = NIL$ ) and ( $k \leq k_{max}$ ) do
5:    $Sol \leftarrow$  YIELDS_iteration( $X, D, C, k, Sol$ )
6:    $k \leftarrow k+1$ 
7:   if !Exceed then
8:     exit
9:   end if
10: end while
11: return  $Sol$ 
```

---



# YIELDS, NG-YIELDS





# YIELDS, NG-YIELDS

- **YI ELDS( $k$ ) :**
  - Variable ordering heuristic based on **previous failures**
- **NG-YI ELDS( $k$ ): Variable ordering heuristic based on no-goods**
  - Take in consideration
    - **Incompatible couples**
    - **Possibly incompatible couples**

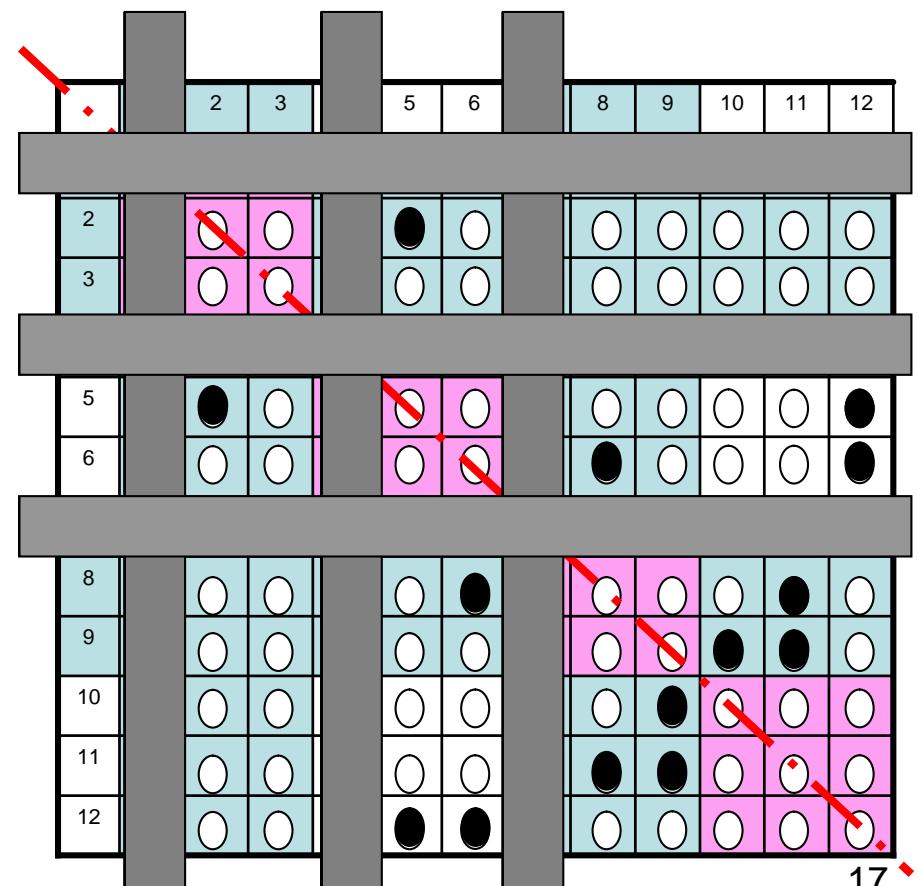
# NG-YIELDS

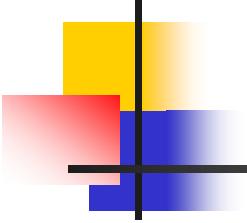
- CSP: Binary matrix
- No-good:

{ $(x_1, 1), (x_2, 4), (x_3, 7)$ }

- {
- $((x_1, 1), (x_2, 4))$
  - $((x_2, 4), (x_3, 7))$
  - $((x_1, 1), (x_3, 7))$

Possibly incompatible  
couples





# NG-YIELDS

- **NG-YIELDS** : Integrate no-good technics into YIELDS method
- **Instantiation couples evolution :**
  - Based on detected **no-goods** (size T)
  - Instantiation couples "possibly" incompatible because they belong to a no-good
- ➔ Couple instantiation matrix: values  $\in [0,1]$   
"Fuzzy matrix"

# Example

	1	2	3	4	5	6	7	8	9	10	11	12
1	○	○	○	●	●	○	○	○	○	○	○	○
2	○	○	○	○	●	○	○	○	○	○	○	○
3	○	○	○	●	○	○	○	○	○	○	○	○
4	●	○	●	●	○	○	○	●	●	○	○	●
5	●	●	○	○	●	●	○	○	○	○	○	●
6	○	○	○	○	○	●	○	●	○	○	○	●
7	●	○	○	●	○	○	○	○	○	●	●	○
8	○	○	○	●	○	●	○	●	○	●	●	○
9	○	○	○	○	○	○	○	○	●	●	●	○
10	○	○	○	○	○	○	●	○	●	○	○	○
11	○	○	○	○	○	●	●	●	○	●	○	○
12	○	○	○	●	●	●	○	○	○	○	○	●

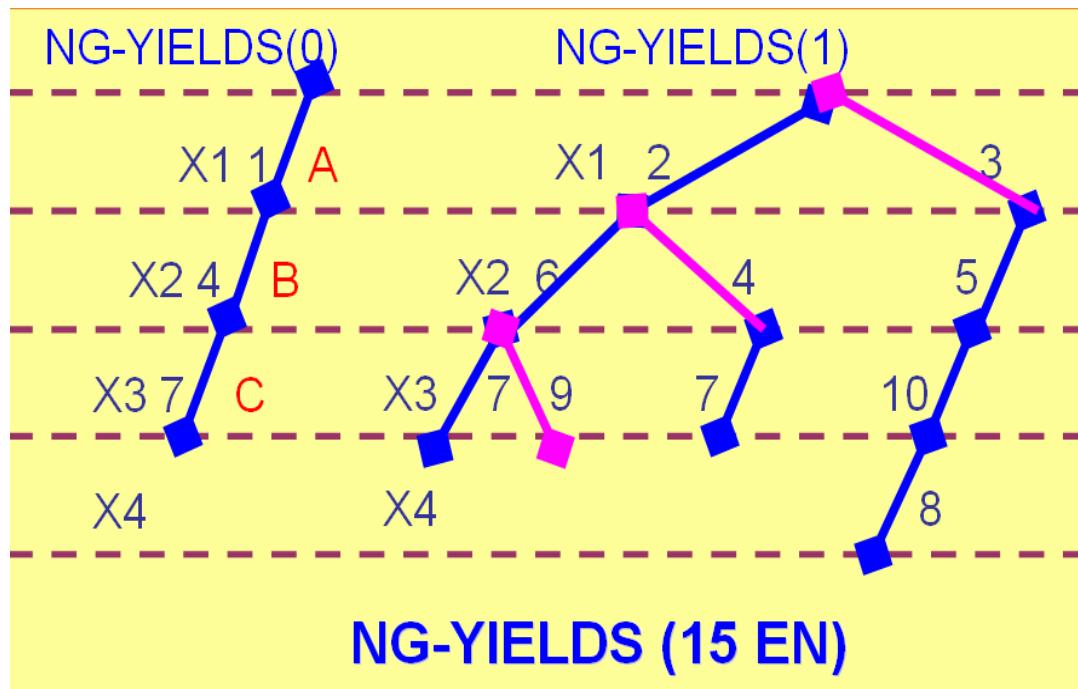
- Compatible instantiation
- Incompatible instantiation
- Possible incompatible instantiation

● =  $\max(\text{previous value}, 2/T)$

Variable ordering heuristic: min\_conflict

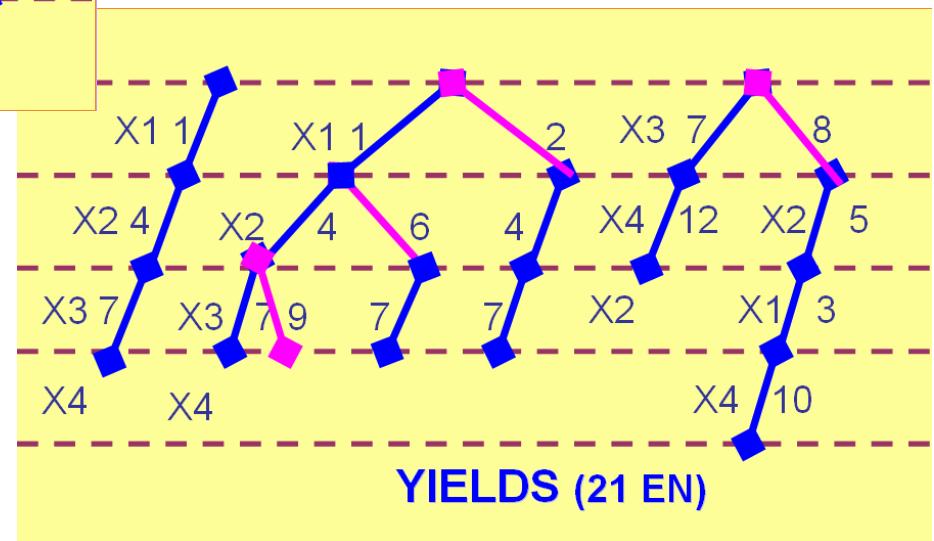
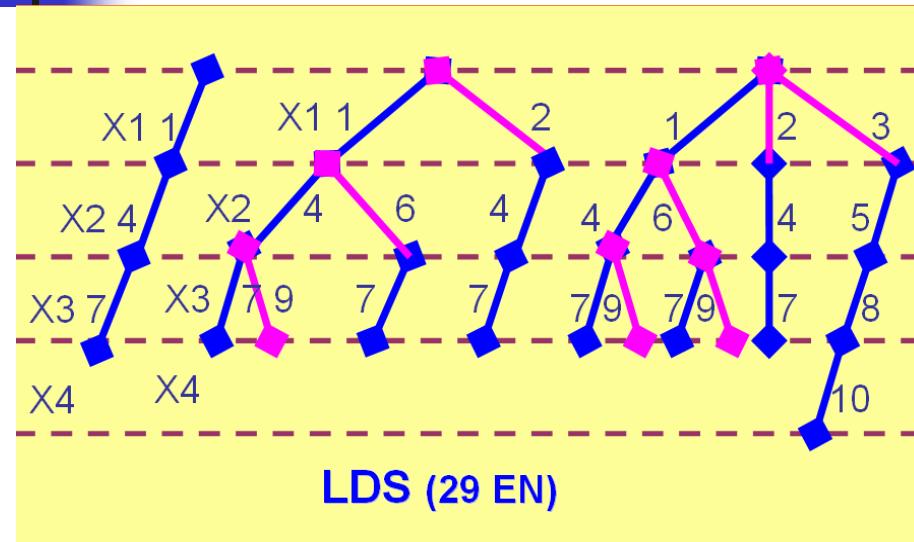
Value ordering heuristic: min\_conflict

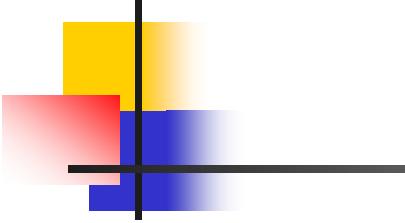
# Example



- ABC = no-good
  - Updates:
    - (A,B)
    - (A,C)
    - (B,C)

# LDS, YIELDS and NG-YIELDs





# NG-YIELDS Algorithm

---

## Algorithm 3 NG-YIELDS(X,D,C,k\_max,Sol)

---

```
1: k  $\leftarrow$  0
2: Sol  $\leftarrow$  NIL
3: Exceed  $\leftarrow$  False
4: tant que (Sol = NIL) and ( $k \leq k_{\text{max}}$ ) faire
5:   Sol  $\leftarrow$  NG-YIELDS_iteration(X,D,C,k,Sol)
6:   k  $\leftarrow$  k+1
7:   si !Exceed alors
8:     exit
9:   fin si
10: fin tant que
11: retourne Sol
```

---

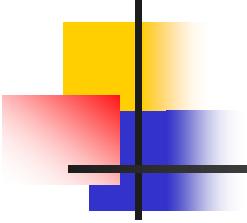
---

## Algorithm 4 NG-YIELDS\_iter(X,D,C,k,Sol)

---

```
1: si X =  $\emptyset$  alors
2:   retourne Sol
3: sinon
4:    $x_i \leftarrow$  First_VariableOrdering(X,Poids)
5:    $v_i \leftarrow$  First_ValueOrdering(Di,k)
6:   si  $v_i \neq \text{NIL}$  alors
7:     D'  $\leftarrow$  Update(X\{xi\},D,C,(xi,vi))
8:     I  $\leftarrow$  NG-YIELDS_iteration(X\{xi\},D',C,k,Sol
       $\cup$  {(xi,vi)})
9:   si I  $\neq$  NIL alors
10:    retourne I
11:   sinon
12:     si k > 0 alors
13:       Di  $\leftarrow$  Di \{vi\}
14:       C  $\leftarrow$  NG-Update(C)
15:       retourne NG-YIELDS_iteration(X,D,C,k-1,Sol)
16:   sinon
17:     Poids[xi]  $\leftarrow$  Poids[xi]+1
18:     Exceed  $\leftarrow$  True // impossible de diverger
19:   fin si
20:   fin si
21:   fin si
22:   retourne NIL
23: fin si
```

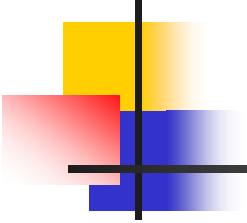
---



# Experimental results

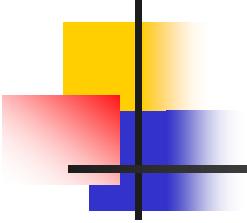
---

- Random CSPs [Gent 96]
- Job-Shop (N. Sadeh) and latin squares (C. Gomes)
- Evaluation vs.:
  - LDS family
  - CSP search methods like CB-AC
- In terms of:
  - CPU time
  - Number of extended nodes (NEN) in the search tree
  - Number of discrepancies needed to reach a solution



# Contribution of YIELDS to the LDS family

- Method characteristics
  - Variable ordering = min\_domain
  - Value ordering = min\_conflict
  - Propagation AC after each instantiation
- Comparison of YIELDS with:
  - LDS/ILDS/DDS
  - Always better (YIELDS>>)

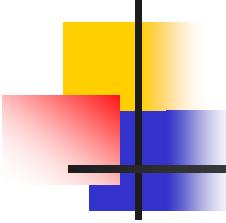


# Contribution of YIELDS to CSP search methods

- Comparison of YIELDS with:
  - CB-AC3 (MAC3\_cache)
- Good results
  - For sparse problems, YIELDS always faster
    - Very good results
  - For dense problems,
    - Always faster but less significant results
    - Isolate tractable problems -> Recover very good results

# Random binary CSPs

Instances	DDS		YIELDS		CB-AC	
	NEN	CPU	NEN	CPU	NEN	CPU
<n,d,C,T>						
<30,25,44,531>(35%sat)	10210510	90.61	6273	0.04	71	0.08
<30,25,44,526>(48%sat)	21876033	207.8	8526	0.06	250	0.19
<30,25,44,518>(73%sat)	1447242	11.72	3543	0.02	270	0.21
<30,25,131,322>(39%sat)	—	—	1342742	12	203862	16.45
<30,25,131,320>(56%sat)	—	—	1360525	11.76	94277	11.92
<30,25,131,318>(74%sat)	—	—	1503581	12.39	54870	13.24
<30,25,131,322>(sat)	—	—	326739	3.07	46583	35
<30,25,131,320>(sat)	—	—	337996	3.05	55165	58.46
<30,25,131,318>(sat)	—	—	341994	3.12	16876	11.87



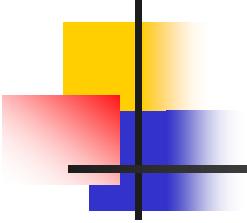
## Job-Shop Instances

# Job-Shop and Latin Squares

instances	YIELDS		CB-AC	
	NEN	CPU	NEN	CPU
enddr1-10-5-1	68997	<1	53186	897
enddr1-10-5-10	57	<1	113486	457
ewddr2-10-5-1	910	<1	92729	480
ewddr2-10-5-10	55	<1	64535	372
e0ddr1-10-5-1	17133261	113	6262916	1752

## Latin squares Instances

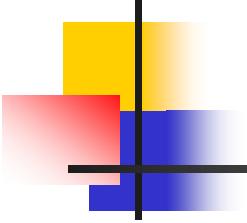
instances	YIELDS		CB-AC	
	NEN	CPU	NEN	CPU
qg.1030	158808	16	68276	72.5
qg.1032	128424	71	80215	105
qg.1034	1775	0.02	181934	212
qg.1036	364	0.01	13609	17.6
qg.1038	114	0.01	14393	18



# Related works

---

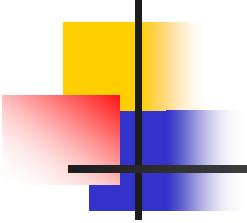
- Squeaky Wheel Optimisation (SWO) [ Joslin & Clements 1998 ]
- Impact-Based Search (IBS) [ Refalo 2004]
- FO-O-Opt (Failure-driven algorithm for Open Hidden-variable Weighted Constraint Optimisation Problems) [ Faltings et al. 2002 ]
- Last Conflict reasoning (LC) [ Boussemart et al. 2004]
- « Restarting LC » [ Grimes & Wallace 2006 ]



# Conclusion

---

- Tree search method to solve CSPs: YIELDS
  - Based on **discrepancies**
  - Principle characteristics:
    - Learning while searching
    - Reduce search tree
- Contribution of YIELDS to the LDS family
  - Less of discrepancies, of time CPU, and of NEN
- Contribution of YIELDS to CSP search methods
  - Better results
- Tested on random and real problems (Job-Shop and latin squares)



# Further works

---

- To evaluate over other benchmarks
- To be compared with some related works
- Learning exploitation by **fuzzy ways**:  
“Semiring-based CSPs”
  - to try other co-norms...
- Incomplete methods for optimisation problems