

**Pruning for the *minimum* Constraint Family
and for the *number of distinct values* Constraint Family**

Nicolas Beldiceanu

SICS

Lägerhyddsvägen 18

75237 Uppsala

nicolas@sics.se

Focus of this Presentation

Existing counting constraints :

- Alldifferent
- Atleast
- Atmost
- Distribute
- Among
- Global cardinality
- Count

A new counting constraint :

- **Nvalue**

Outline of the Presentation

The *minimum* and *number of distinct values* Families

- The *minimum* Family
- The *number of distinct values* Family

Filtering Algorithms

- Organisation of the Algorithms
- Required Services
- Lower Bound for the Minimum Number of Distinct Values
- Pruning According to the Maximum Number of Distinct Values

Conclusion

The *minimum* and *number of distinct values* Families

- ➔ • The *minimum* Family
- The *number of distinct values* Family

Filtering Algorithms

- Organisation of the Algorithms
- Required Services
- Lower Bound for the Minimum Number of Distinct Values
- Pruning According to the Maximum Number of Distinct Values

Conclusion

I
N
S
T
A
N
C
E
S

Equivalence relation	Ordering relation	Member and example of solution
$X_1 = X_2$	$X_1 < X_2$	minimum($M, \{V_1, \dots, V_n\}$) minimum(3 , {4,4, 3 ,4,6})
$X_1 = X_2$	$X_1 > X_2$	maximum($M, \{V_1, \dots, V_n\}$) maximum(6 , {4,4, 3 ,4,6})
$X_1 = X_2$	$X_1 < X_2$	min_n($M, r, \{V_1, \dots, V_n\}$) min_n(3 ,2,{1,8,1,5, 3 ,8})
$X_1 = X_2$	$X_1 < X_2$	max_n($M, r, \{V_1, \dots, V_n\}$) max_n(5 ,2,{1,8,1, 5 ,3,8})
$X_1 = X_2 \wedge$ $Y_1 = Y_2$	$X_1 < X_2 \vee$ $(X_1 = X_2 \wedge Y_1 < Y_2)$	minimum_pair($M, \{P_1, \dots, P_n\}$) minimum_pair(3-2 , {5-6, 3-7, 3-2 , 4-1})

The *minimum* and *number of distinct values* Families

- The *minimum* Family

- ➔ • The *number of distinct values* Family

Filtering Algorithms

- Organisation of the Algorithms
- Required Services
- Lower Bound for the Minimum Number of Distinct Values
- Pruning According to the Maximum Number of Distinct Values

Conclusion

The *nvalue* Family

Classical **counting** constraints:

- **among** : CHIP
- **count** : SICStus
- **distribute** : Ilog Solver

```
among( 3,  
       {var-5,var-8,var-5,var-  
5},  
       {val-5})
```

Differences with classical counting constraints:

- Does not count **several times** the same value
- Can't have **complete pruning** in polynomial time (*domination in graph*)

```
nvalue( 2,  
        {var-5,var-8,var-5,var- 5})
```

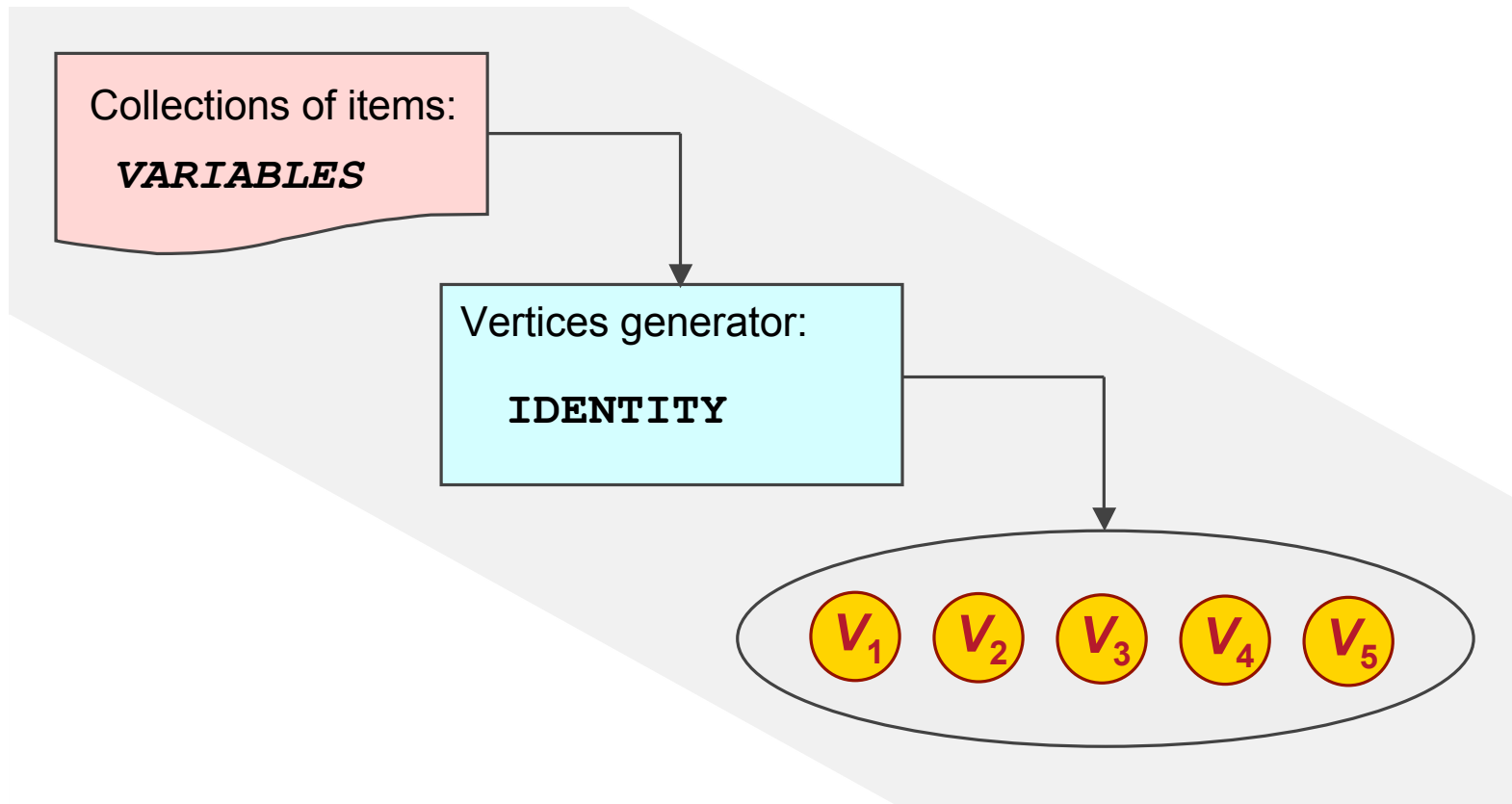
nvalue(NVAL, VARIABLES)

• ARGUMENT	:	NVAL	:	dvar	
					VARIABLES: collection(var-dvar)
• RESTRICTION(S)	:	NVAL ≥ 0			
					NVAL ≤ VARIABLES
					required(VARIABLES.var)
• VERTEX INPUT	:	VARIABLES			
• VERTEX GENERATOR	:	IDENTITY			
• EDGE INPUT	:	VARIABLES			
• EDGE GENERATOR	:	CLIQUE			
• EDGE ARITY	:	2			
• EDGE CONSTRAINT	:	VARIABLES.var[1] = VARIABLES.var[2]			
• GRAPH PROPERTY	:	NSCC = NVAL			

nvalue(4, { var-3, var-1, var-7, var-1, var-6 })

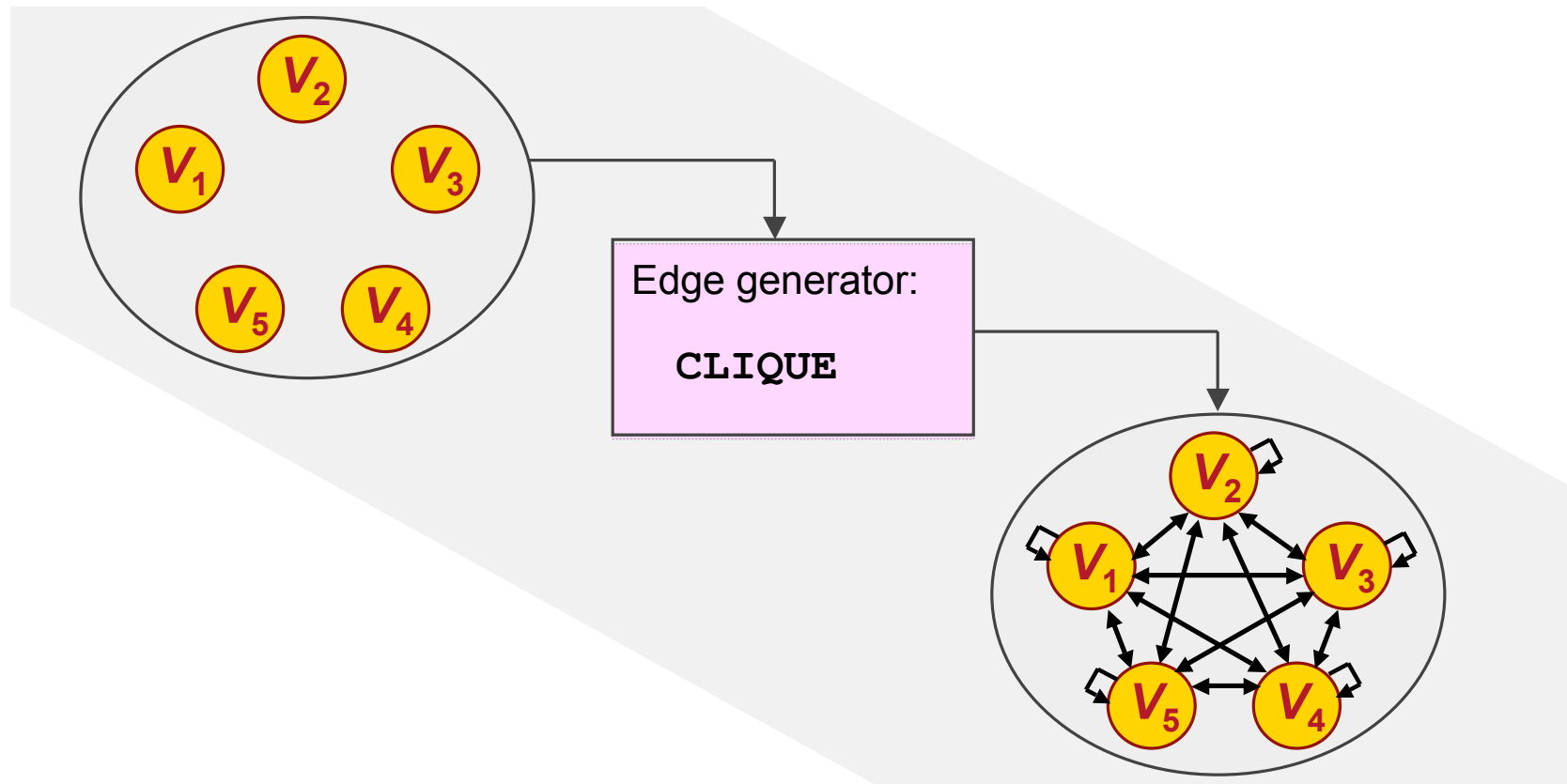
Graph Generation for $nvalue(NVAL, VARIABLES)$

- VERTEX INPUT : *VARIABLES*
- VERTEX GENERATOR : IDENTITY



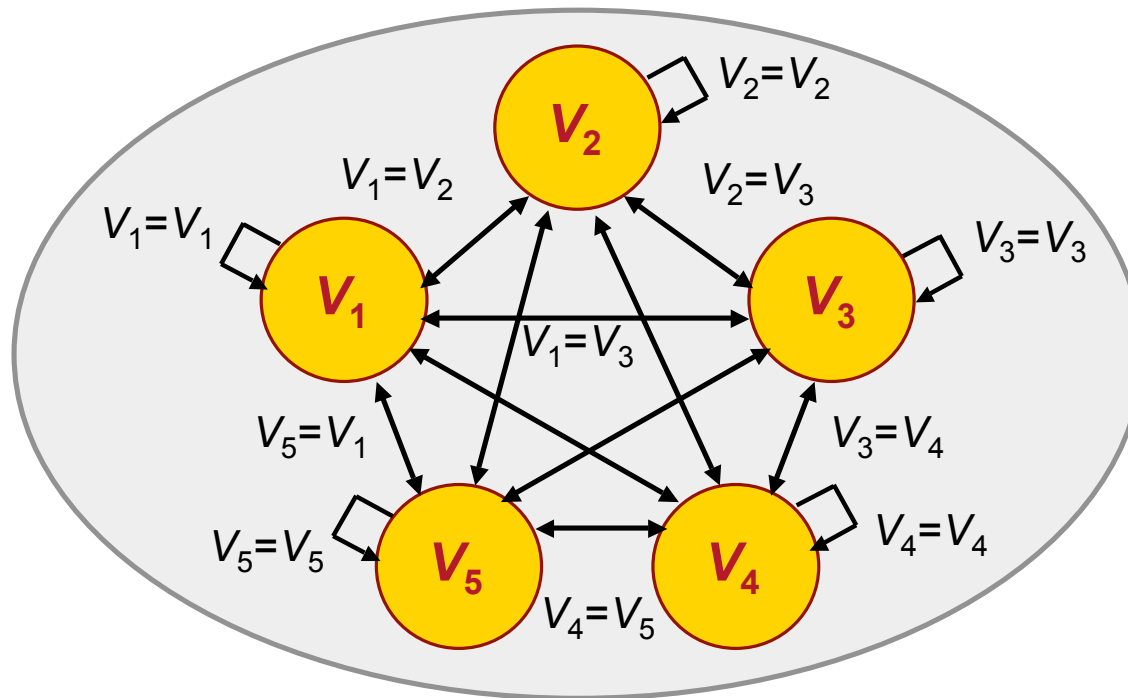
Graph Generation for $nvalue(NVAL, VARIABLES)$

- **EDGE INPUT** : *VARIABLES*
- **EDGE GENERATOR** : **CLIQUE**
- **EDGE ARITY** : 2



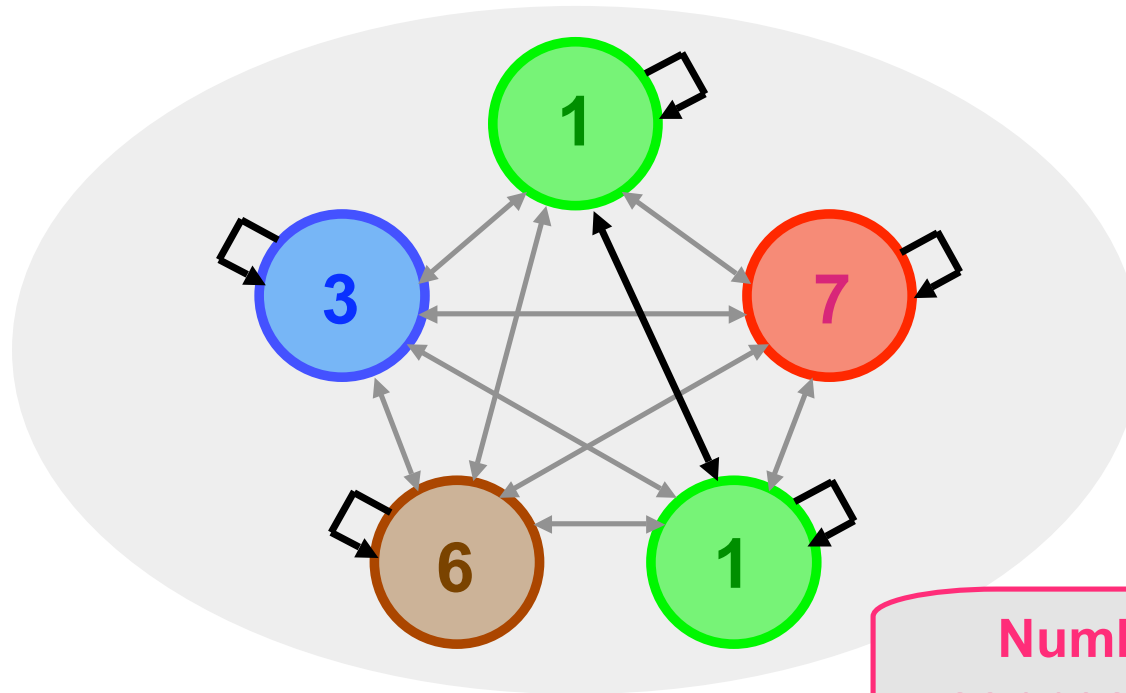
Elementary Constraint for $nvalue(NVAL, VARIABLES)$

- **EDGE CONSTRAINT:** $VARIABLES.var[1] = VARIABLE.var[2]$



Graph Property for $nvalue(NVAL, VARIABLES)$

- GRAPH PROPERTY : NSCC = NVAL



Number of strongly connected components

$nvalue(4, \{ var-3, var-1, var-7, var-1, var-6 \})$

"Signature" of the *number of distinct values* Family

• VERTEX	GENERATOR :	IDENTITY
• EDGE	GENERATOR :	CLIQUE
• EDGE	ARITY :	2
• CONSTRAINT	PROPERTY :	equivalence
• GRAPH	PROPERTY :	NSCC = NVAL

The *minimum* and *number of distinct values* Families

- The *minimum* Family
- The *number of distinct values* Family

Filtering Algorithms

- ➔ • **Organisation of the Algorithms**
- Required Services
- Lower Bound for the Minimum Number of Distinct Values
- Pruning According to the Maximum Number of Distinct Values

Conclusion

Organisation of the Algorithms

nvalue(**Nvalue**, {**Var**₁, ..., **Var**_n })

FILTERING ALGORITHMS

find **BOUNDS**
for **Nvalue**

PROPAGATE from bounds
of **Nvalue** to {**Var**₁, ..., **Var**_n }

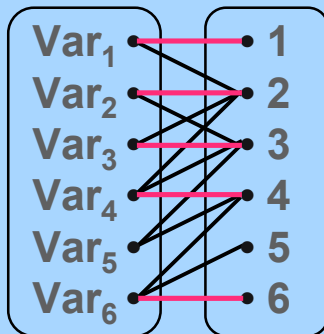
Organisation of the Algorithms :
maximum number of distincts values

nvalue(5..6, {1..2, 2..3, 2..3, 2..4, 3..4, 4..6})

find **BOUNDS**
for **Nvalue**

PROPAGATE from bounds
of **Nvalue** to $\{\text{Var}_1, \dots, \text{Var}_n\}$

$\max(\text{Nvalue}) \leq 5$



nvalue(5, {1..2, 2..3, 2..3, 2..4, 3..4, 4..6})

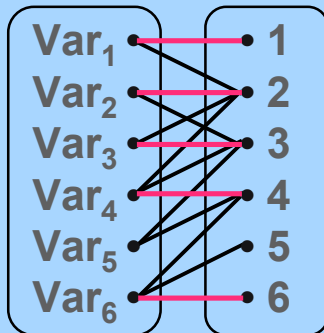
Organisation of the Algorithms :
 maximum number of distinct values

nvalue(5..6, {1..2, 2..3, 2..3, 2..4, 3..4, 4..6})

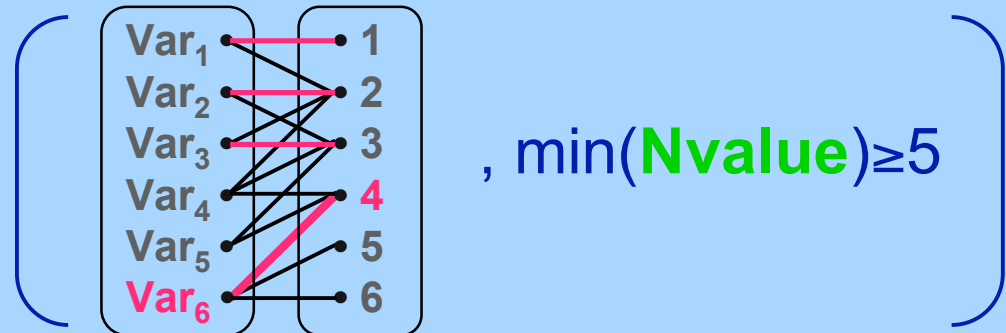
find **BOUNDS**
 for **Nvalue**

PROPAGATE from bounds
 of **Nvalue** to $\{\text{Var}_1, \dots, \text{Var}_n\}$

max(Nvalue) ≤ 5



Var₆ ≠ 4



nvalue(5, {1, 2..3, 2..3, 2..4, 3..4, 5..6})

Organisation of the Algorithms :
minimum number of distincts values

nvalue(**0..2**, {1..2, 2..3, 2..3, 2..4, 3..4, 4..6})

find **BOUNDS**
for **Nvalue**

PROPAGATE from bounds
of **Nvalue** to $\{\text{Var}_1, \dots, \text{Var}_n\}$



min(Nvalue) ≥ 2
(1..2 \cap 4..6 = \emptyset)

nvalue(**2**, {1..2, 2..3, 2..3, 2..4, 3..4, 4..6})

Organisation of the Algorithms :
minimum number of distincts values

nvalue(**0..2**, {1..2, 2..3, 2..3, 2..4, 3..4, 4..6})

find **BOUNDS**
for **Nvalue**

PROPAGATE from bounds
of **Nvalue** to $\{\text{Var}_1, \dots, \text{Var}_n\}$

min(Nvalue) ≥ 2
(1..2 \cap 4..6 = \emptyset)

Var₆ ≠ 6
(1..2 \cap 3..4 = \emptyset , 1..2 \cap 6 = \emptyset ,
6 \cap 3..4 = \emptyset , max(**Nvalue**) ≤ 2)

nvalue(**2**, {2, 2, 2, [2,4], 4, 4})

Organisation of the Algorithms

- Evaluate max(Nvalue)
 - Prune according to min(Nvalue)
- } : similar to *alldifferent*

- Evaluate min(Nvalue)
 - Prune according to max(Nvalue)
- } : **focus on this part**

The *minimum* and *number of distinct values* Families

- The *minimum* Family
- The *number of distinct values* Family

Filtering Algorithms

- Organisation of the Algorithms
- ➔ • **Required Services**
- Lower Bound for the Minimum Number of Distinct Values
- Pruning According to the Maximum Number of Distinct Values

Conclusion

Required Services

Access to an
equivalence
relation with a
total ordering
among classes

- **min_item**: returns smallest class
- **max_item**: returns largest class
- $I < J$: true iff class I less than class J
- $I > J$: true iff class I less than class J
- **next(I)** : if $I \neq \text{max_item}$ then smallest $J > I$
- **prev(I)** : if $I \neq \text{min_item}$ then largest $J < I$

Required Services

**Domain
variables and
equivalence
relation**

- **min(V)** : minimum class $\in \text{dom}(V)$
- **max(V)** : maximum class $\in \text{dom}(V)$
- **remove_val(V,I)** removes items \in class I from V
- **adjust_min(V,I)**: removes items \in class $J < I$ from V
- **adjust_max(V,I)** removes items \in class $J > I$ from V

The *minimum* and *number of distinct values* Families

- The *minimum* Family
- The *number of distinct values* Family

Filtering Algorithms

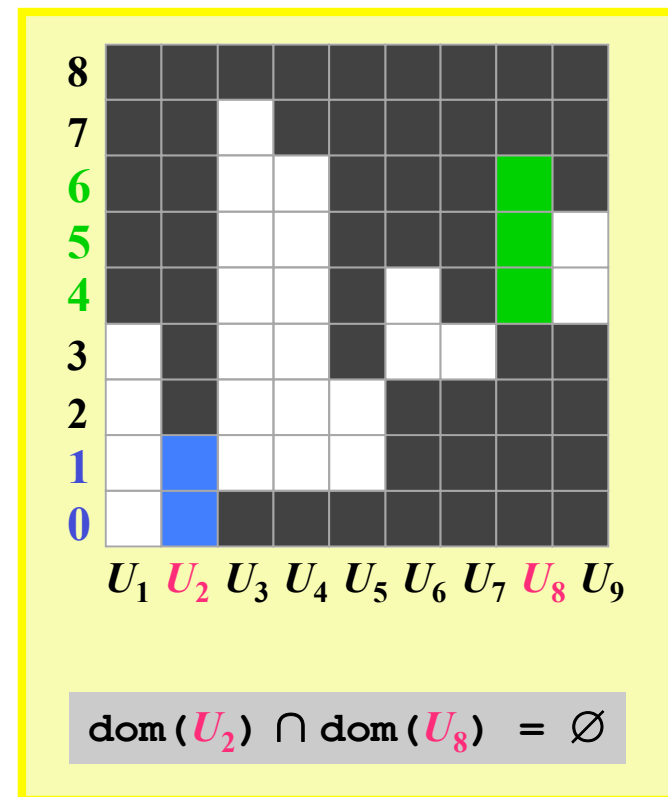
- Organisation of the Algorithms
- Required Services
- ➔ • **Lower Bound for the Minimum Number of Distinct Values**
- Pruning According to the Maximum Number of Distinct Values

Conclusion

Computing the Minimum Number of Distinct Values

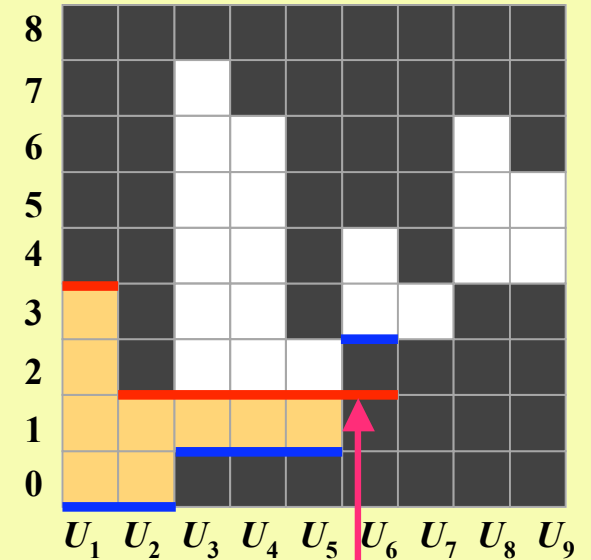
IDEA

Find sets of variables for which the domains **do not** pairwise intersect.



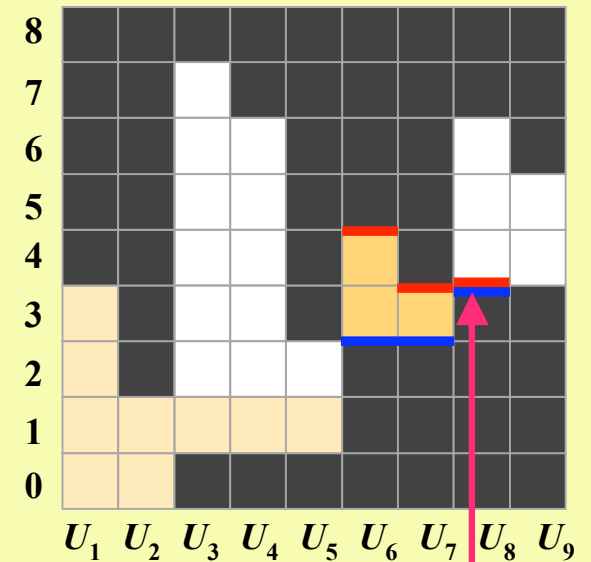
Computing the Minimum Number of Distinct Values

```
1 U1,...,Un:=sort V1,...,Vn on increasing minimum value;  
2 ndistinct:=1;  
3 reinit:=1;  
4 i:=1;  
5 WHILE i<n DO  
6   i:=i+1-reinit;  
7   IF reinit OR low<min(Ui) THEN low:=min(Ui) ENDIF;  
8   IF reinit OR up >max(Ui) THEN up :=max(Ui) ENDIF;  
9   reinit:=(low>up);  
10  ndistinct:=ndistinct+reinit;  
11 ENDWHILE;
```



Computing the Minimum Number of Distinct Values

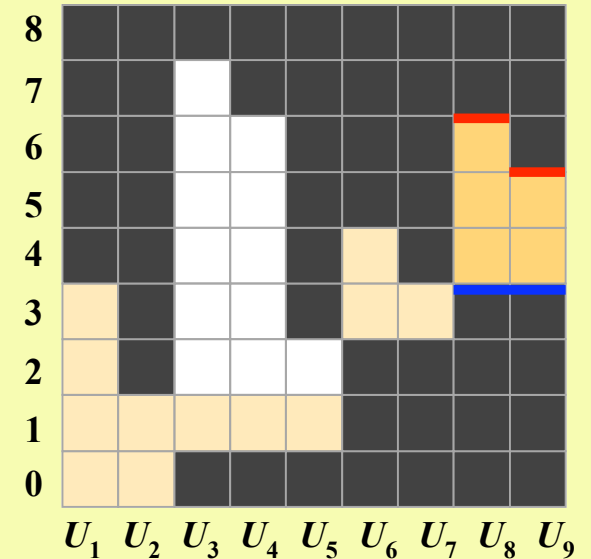
```
1 U1,...,Un:=sort V1,...,Vn on increasing minimum value;  
2 ndistinct:=1;  
3 reinit:=1;  
4 i:=1;  
5 WHILE i<n DO  
6   i:=i+1-reinit;  
7   IF reinit OR low<min(Ui) THEN low:=min(Ui) ENDIF;  
8   IF reinit OR up >max(Ui) THEN up :=max(Ui) ENDIF;  
9   reinit:=(low>up);  
10  ndistinct:=ndistinct+reinit;  
11 ENDWHILE;
```



```
reinit:=(4>3)  
ndistinct:=2+1
```

Computing the Minimum Number of Distinct Values

```
1 U1,...,Un:=sort V1,...,Vn on increasing minimum value;
2 ndistinct:=1;
3 reinit:=1;
4 i:=1;
5 WHILE i<n DO
6   i:=i+1-reinit;
7   IF reinit OR low<min(Ui) THEN low:=min(Ui) ENDIF;
8   IF reinit OR up >max(Ui) THEN up :=max(Ui) ENDIF;
9   reinit:=(low>up);
10  ndistinct:=ndistinct+reinit;
11 ENDWHILE;
```



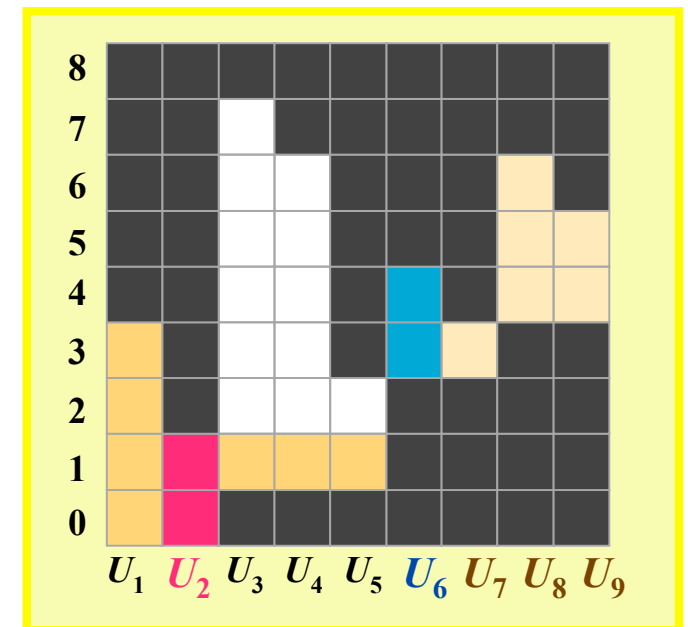
We have at least **3 distinct values** for variables U_1, \dots, U_9

Validity of the Lower Bound

```
1 reinit=1
2 U1 reinit=0 U2 reinit=0 U3 reinit=0 U4 reinit=0 U5 reinit=0 U6 reinit=1
3 U6 reinit=0 U7 reinit=0 U8 reinit=1
4 U8 reinit=0 U9 reinit=0
```

- Build groups of consecutive variables: $U_1 U_2 U_3 U_4 U_5$, $U_6 U_7$, $U_8 U_9$
- In a group consider the variable with smallest maximum value M (U_2 for group 1)
- The min.of the first variable of the next group is greater than M ($\min(U_6) > \max(U_2)$)
- Since the minimum of the next variables U_7, U_8, U_9 are greater or equal then $\min(U_6)$:

We found in group 1 a variable which does not intersect all the next variables



Validity of the Lower Bound

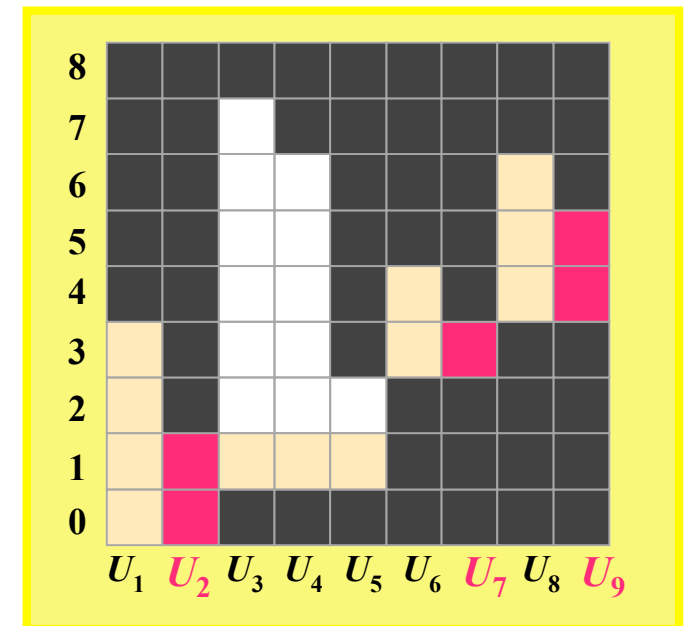
```
1 reinit=1
2 U1 reinit=0 U2 reinit=0 U3 reinit=0 U4 reinit=0 U5 reinit=0 U6 reinit=1
3 U6 reinit=0 U7 reinit=0 U8 reinit=1
4 U8 reinit=0 U9 reinit=0
```

- Repeat the same process for each group:
 - ◆ Group 1: U_2 does not intersect U_6, U_7, U_8, U_9
 - ◆ Group 2: U_7 does not intersect U_8, U_9
 - ◆ Group 3: U_9

We found $n_{distinct} = 3$ variables
which do not pairwise intersect

⇒

at least 3 distinct values



Tightness of the Lower Bound

If for **each group** of variables there is at least **one value in common**, then the lower bound is **sharp**.

The *minimum* and *number of distinct values* Families

- The *minimum* Family
- The *number of distinct values* Family

Filtering Algorithms

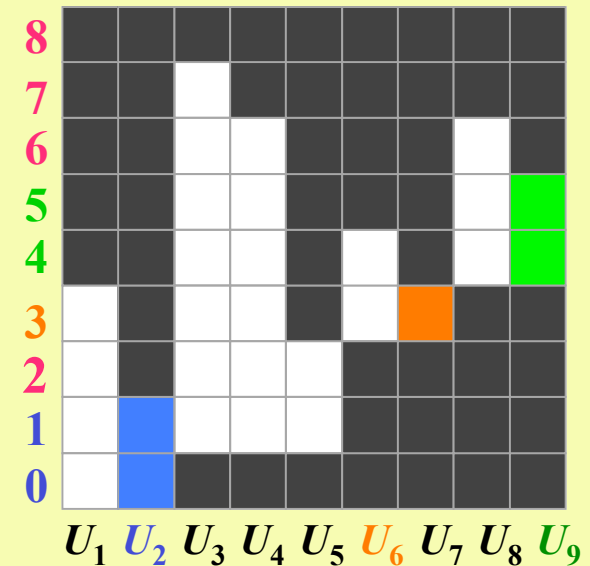
- Organisation of the Algorithms
- Required Services
- Lower Bound for the Minimum Number of Distinct Values
- ➔ • **Pruning According to the Maximum Number of Distinct Values**

Conclusion

Pruning According to the Maximum Number of Distinct Values

IDEA

All values **outside** the domains of a set of variables for which the domains do not pairwise intersect lead to **one extra value**.



$$\text{dom}(U_2) \cap \text{dom}(U_7) = \emptyset$$

$$\text{dom}(U_2) \cap \text{dom}(U_9) = \emptyset$$

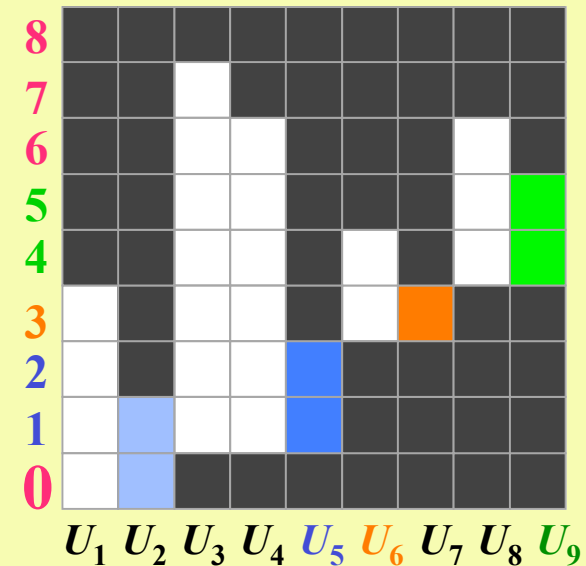
$$\text{dom}(U_7) \cap \text{dom}(U_9) = \emptyset$$

If lower bound=3 and nvalue≤3:
don't use 2,6,7,8

Pruning According to the Maximum Number of Distinct Values

PROBLEM

The pruning **may depend** on the set of variables we consider.



$$\text{dom}(U_5) \cap \text{dom}(U_7) = \emptyset$$

$$\text{dom}(U_5) \cap \text{dom}(U_9) = \emptyset$$

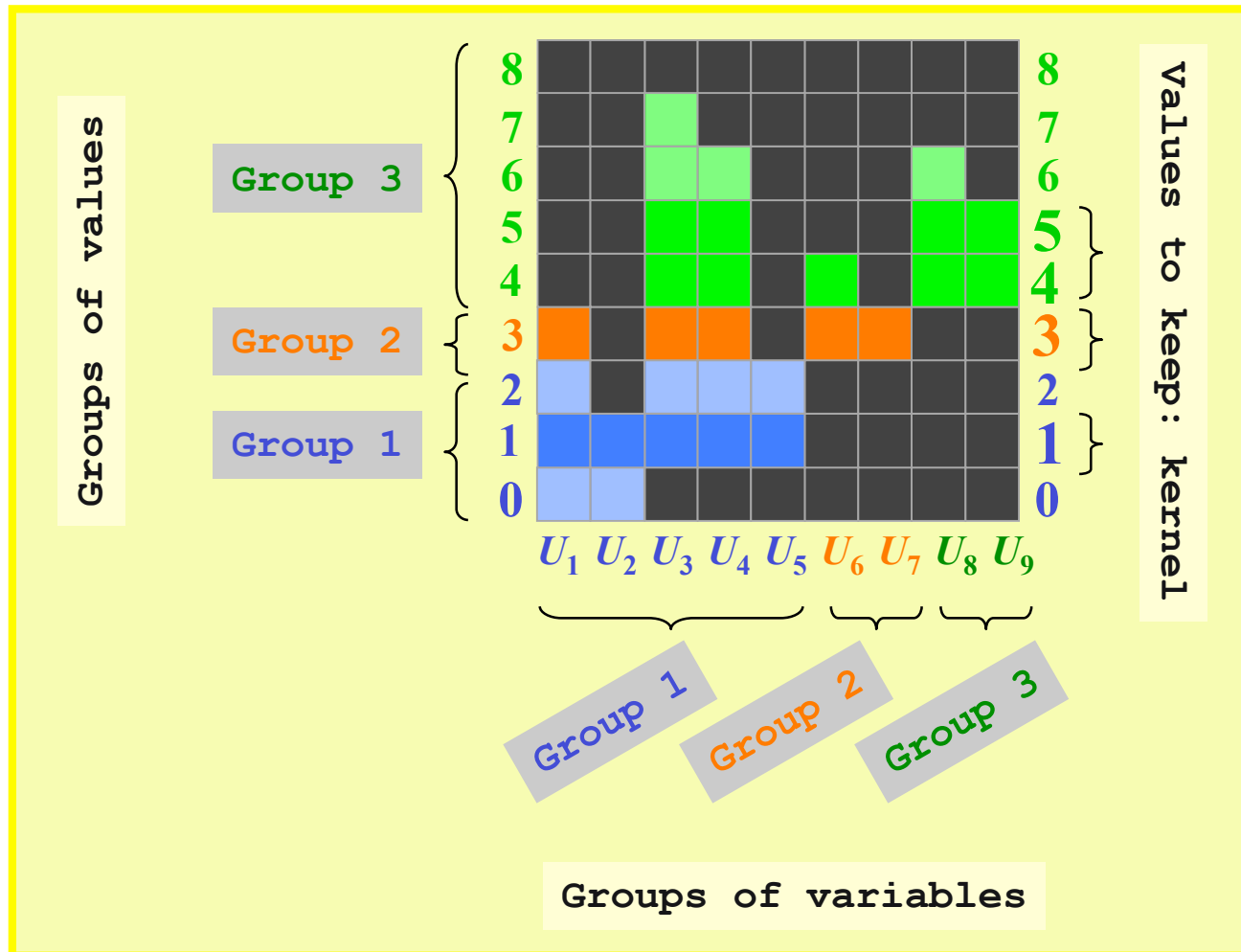
$$\text{dom}(U_7) \cap \text{dom}(U_9) = \emptyset$$

If lower bound=3 and nvalue≤3:

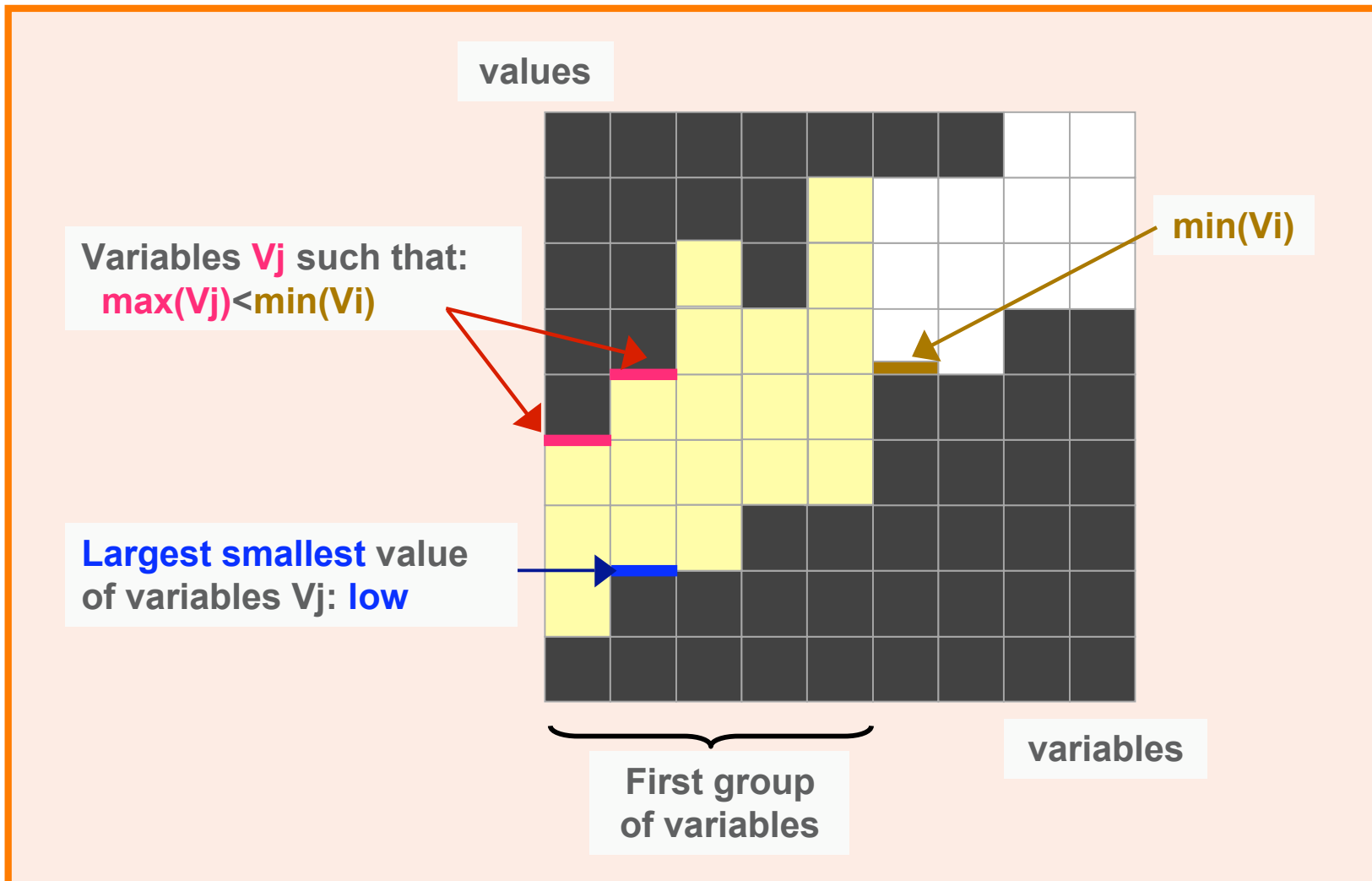
don't use 0,6,7,8

Pruning According to the Maximum Number of Distinct Values

IDEA



Computing the **Minimum** of the Kernel of a Group

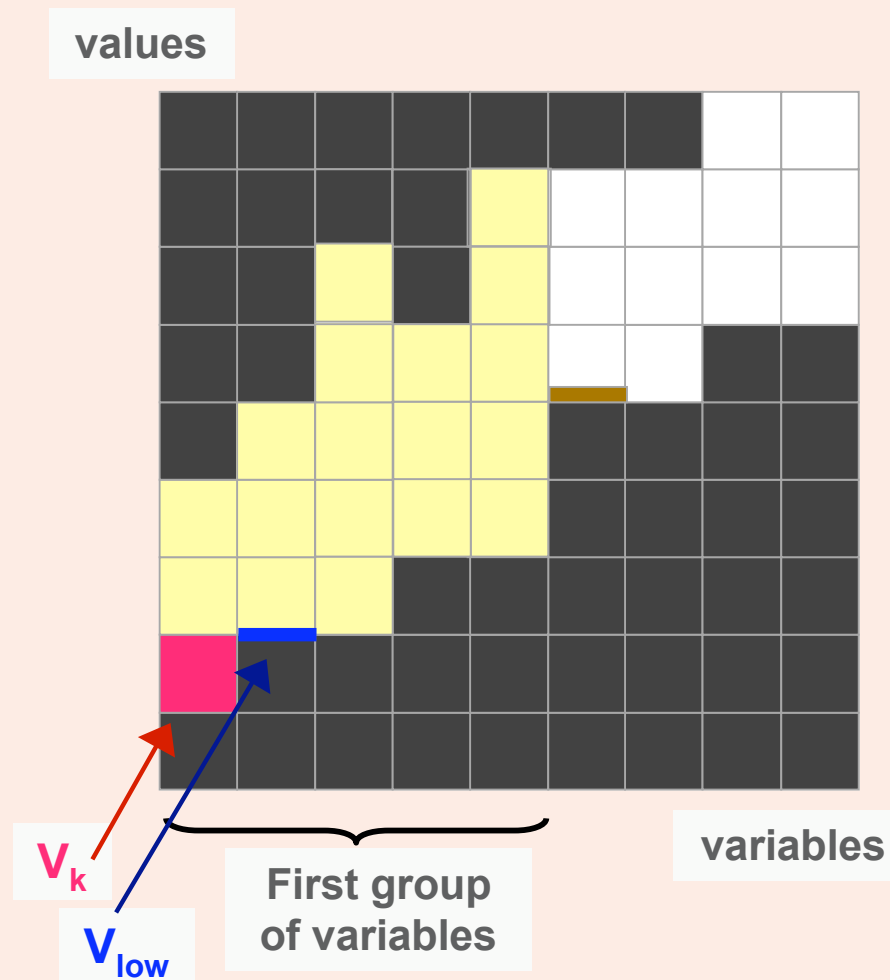


Pruning According to the Minimum of the Kernel of a Group

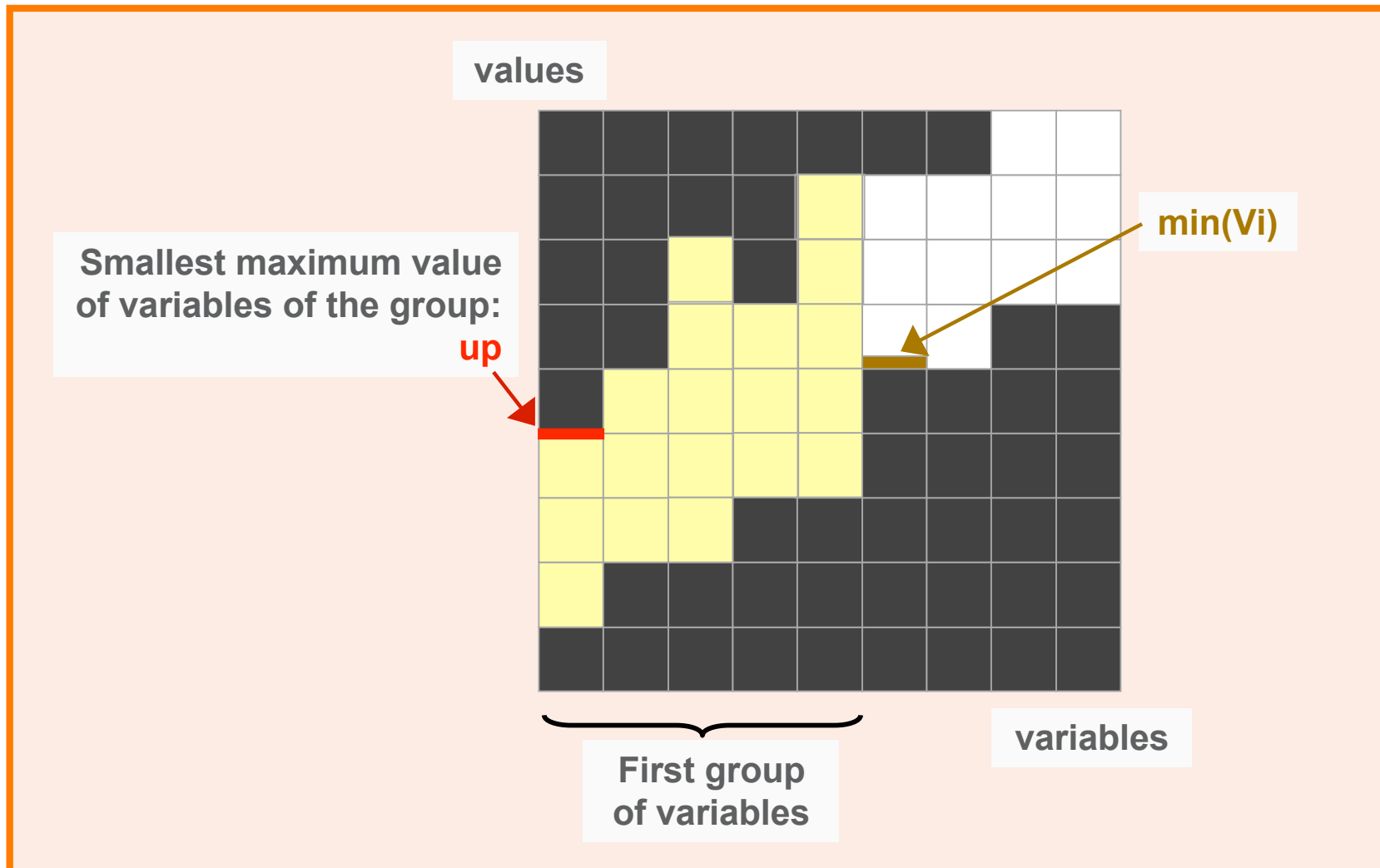
ASSUME that a variable V_k takes a value $< \min(V_{low})$:

(1) Value of V_k can't be taken by variables of other groups

(2) One extra value in the group since V_{low} can't take the value of V_k



Computing the **Maximum** of the Kernel of a Group

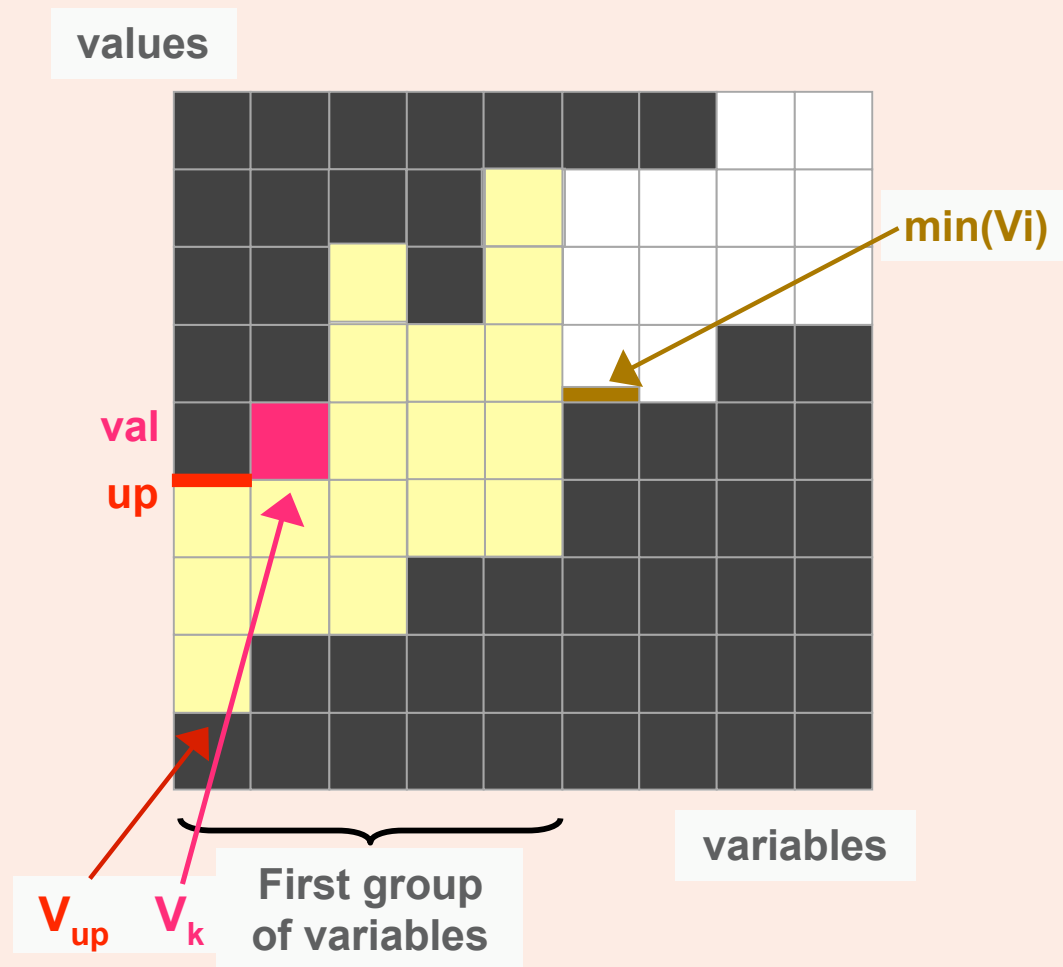


Pruning According to the Maximum of the Kernel of a Group

ASSUME that a variable V_k of the group takes a value val :
 $up < val < \min(V_i)$

(1) Value val of V_k can't be taken by variables of the other groups

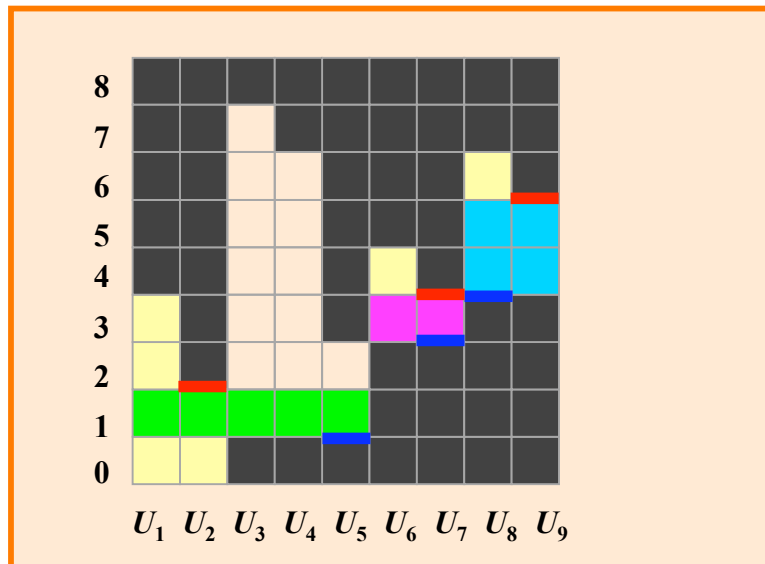
(2) One extra value in first group since V_{up} can't take the value val of V_k



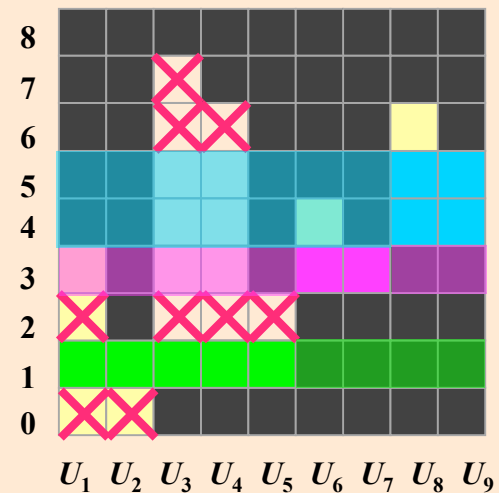
Pruning According to the Maximum Number of Distinct Values

If no more than 3 distinct values then:

Discard all values that do not belong to the union of the kernels



Computing the kernel of each group
(*linear*)



Pruning according to the kernels
(*proportional to number of removed values*)

Conclusion

What was achieved :

- A **generic** filtering algorithm for **two families** of constraints

What could be done :

- Take partially into account **holes** of the domains
- Specific algorithm for **small** values of r (for minimum)

minimum(*MIN*, *VARIABLES*)

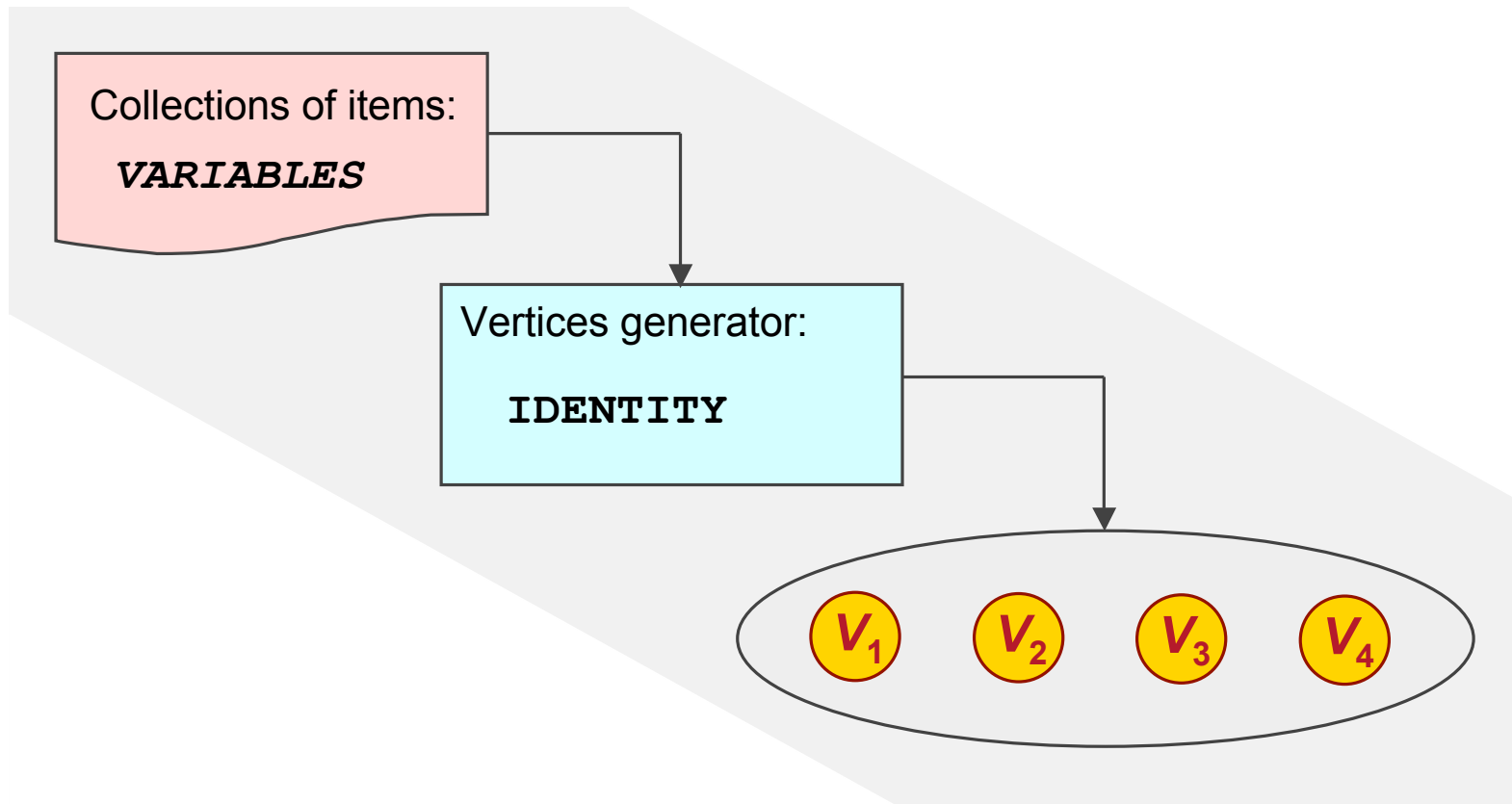
• ARGUMENT	:	<i>MIN</i>	:	dvar
				<i>VARIABLES</i> : collection(<i>var</i> -dvar)
• RESTRICTION(S)	:	required(<i>VARIABLES.var</i>)		
• VERTEX INPUT	:	<i>VARIABLES</i>		
• VERTEX GENERATOR	:	IDENTITY		
• EDGE INPUT	:	<i>VARIABLES</i>		
• EDGE GENERATOR	:	CLIQUE		
• EDGE ARITY	:	2		
• EDGE CONSTRAINT	:	<i>VARIABLES</i> [1]= <i>VARIABLES</i> [2]* v <i>VARIABLES.var</i> [1] < <i>VARIABLES.var</i> [2]		
• GRAPH PROPERTY	:	ORDER(0,maxint, <i>var</i>) = <i>MIN</i>		

* True is 1 and 2 corresponds to the same node

minimum(**3**, { *var-5*,*var-7*,*var-3*,*var-6* })

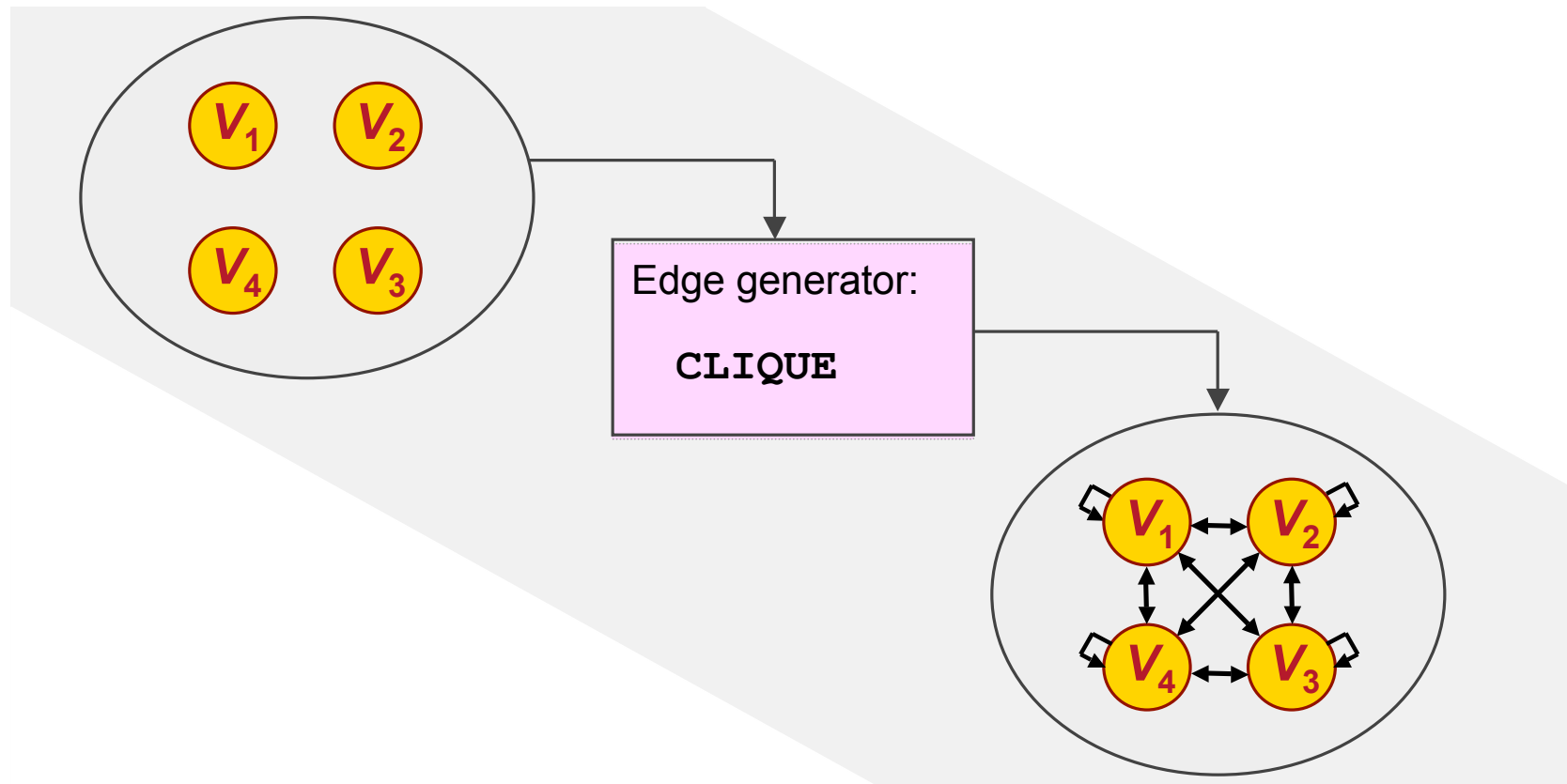
Graph Generation for $\text{minimum}(MIN, VARIABLES)$

- VERTEX INPUT : *VARIABLES*
- VERTEX GENERATOR : IDENTITY



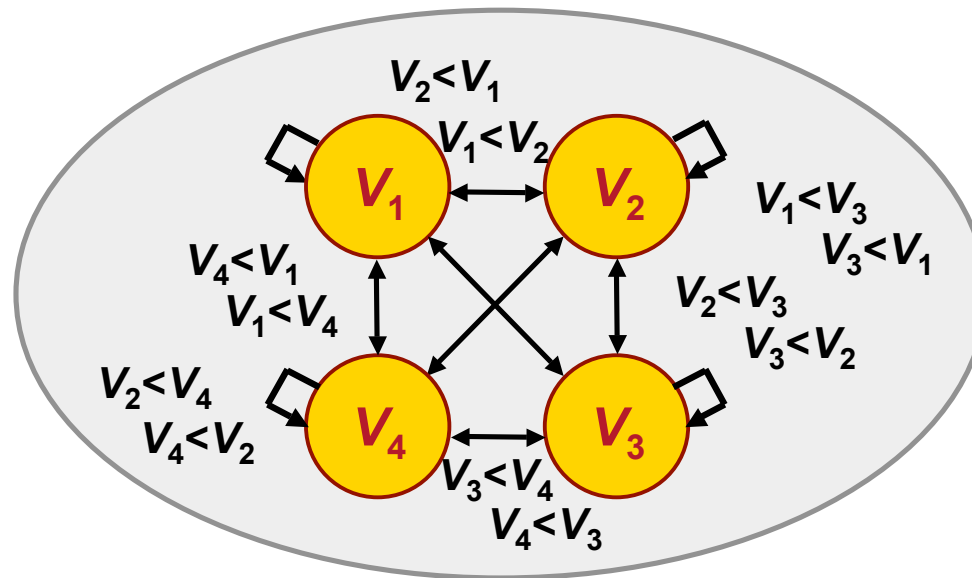
Graph Generation for minimum(*MIN*,*VARIABLES*)

- **EDGE INPUT** : *VARIABLES*
- **EDGE GENERATOR** : **CLIQUE**
- **EDGE ARITY** : 2



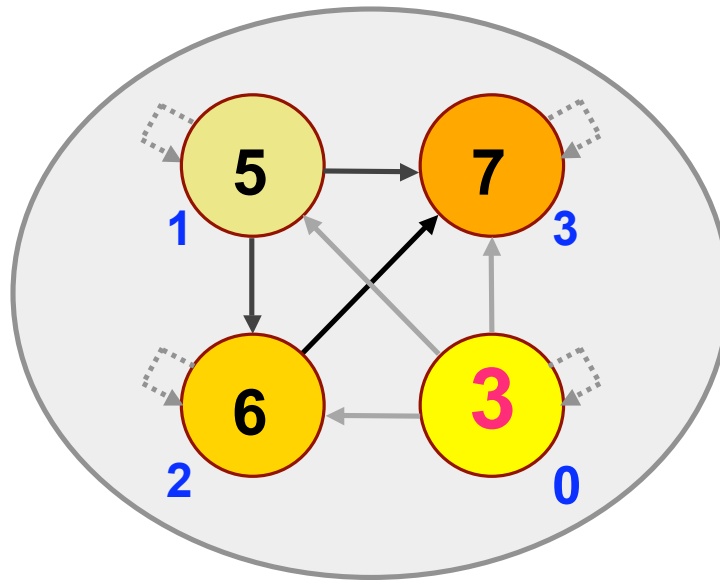
Elementary Constraint for minimum(*MIN*,*VARIABLES*)

- EDGE CONSTRAINT: $VARIABLES[1]=VARIABLES[2] \vee$
 $VARIABLES.var[1] < VARIABLES.var[2]$



Graph Property for minimum(*MIN*,*VARIABLES*)

- GRAPH PROPERTY : ORDER(0,maxint,var) = *MIN*



minimum(3, { var-5,var-7,var-3,var-6 })

"Signature" of the *minimum* Family

• VERTEX	GENERATOR :	IDENTITY
• EDGE	GENERATOR :	CLIQUE
• EDGE	ARITY :	2
• CONSTRAINT	PROPERTY :	total order over equivalence classes
• GRAPH	PROPERTY :	ORDER(<i>rank, default*</i> , <i>attribute</i>) = VAR

**default* is largest represent of the equivalence class

A Model for *nvalue* Using the *global cardinality* Constraint

Nvalue in 1..4

[Cpt₁, Cpt₂, Cpt₃] in 0..4

[Remainder₁, Remainder₂, Remainder₃] in 0..3

[**Var**₁, **Var**₂, **Var**₃, **Var**₄] in 0..2

[Bool₁, Bool₂, Bool₃] in 0..1

```
global_cardinality( {var-Var1, var-Var2, var-Var3, var-Var4},  
                   {var-Cpt1 val-0, var-Cpt2 val-1, var-Cpt3 val-2} )
```

$$\text{Cpt}_1 + 3 = 4 \cdot \text{Bool}_1 + \text{Remainder}_1$$

$$\text{Cpt}_2 + 3 = 4 \cdot \text{Bool}_2 + \text{Remainder}_2$$

$$\text{Cpt}_3 + 3 = 4 \cdot \text{Bool}_3 + \text{Remainder}_3$$

$$\text{Nvalue} = \text{Bool}_1 + \text{Bool}_2 + \text{Bool}_3$$

```
nvalue( Nvalue,  
        {var-Var1, var-Var2,  
          var-Var3, var-Var4} )
```

A Model for *nvalue* Using the *global cardinality* Constraint

Nvalue in 1..4

[**Cpt**₁, **Cpt**₂, **Cpt**₃] in 0..4

[**Remainder**₁, **Remainder**₂, **Remainder**₃] in 0..3

[**Var**₁, **Var**₂, **Var**₃, **Var**₄] in 0..2

[**Bool**₁, **Bool**₂, **Bool**₃] in 0..1

global_cardinality({*var-0*, *var-2*, *var-0*, *var-0*},
 {*var-3 val-0*, *var-0 val-1*, *var-1 val-2*})

$$3 + 3 = 4.1 + 2$$

$$0 + 3 = 4.0 + 3$$

$$1 + 3 = 4.1 + 0$$

$$2 = 1 + 0 + 1$$

nvalue(2,
 {*var-0*, *var-2*,
 var-0, *var-0*})