# Etude préliminaire sur les symétries, leur élimination par la méthode SBDS et le apports possibles de la visualisation

Pierre Deransart Guillaume Arnaud* Abderrahmane Aggoun†

**Résumé**

La première partie pose les fondements théoriques de l'utilisations de vues pour détecter des relations entre éléments d'un ensemble, quand les relations sont des groupes de bijections. On suppose que l'on a une collection d'objets qui sont liés par des relations dont on ne connaît pas a priori la nature. Ces relations sont potentiellement révélées par des vues appropriées. Quand une partie des relations possible est connue, on suppose que l'on est en mesure de les "casser", c'est à dire de produire de nouvelles collections d'objets dans les relations desquels les relations "cassées" ont disparu. On se pose alors le problème de savoir ce que l'on est susceptible d'observer dans les vues qui peuvent être construites avec les objets restants. Le but est d'appliquer ce travail à la détection de symétries dans les CSP à partir de techniques de visualisation.

La deuxième partie propose une formalisation commune sur les symétries, pouvant être utilisée dans le cadre des algorithmes d'éliminations, de visualisations et de détection. Une implantation de la SBDS est également proposée en GNU Prolog ainsi que des propositions de vue pour la détection de symétries.

Enfin la dernière partie donne une possibilité de vue pour la détection de symétries basée sur une heuristique portant sur le rapport entre le couple (variable,valeur) et le nombre de solutions. Bien qu'a posteriori, elle peut s'avérer utile pour détecter des symétries compliquées.

**Keywords :** CSP, symétrie, fonction préservante, super-groupes

*INRIA-Rocquencourt, BP105, 78153 Le Chesnay Cedex, France, phone +33 (0)1 39635097 fax +33 (0)1 39634569, email : {pierre.deransart, guillaume.arnaud}inria.fr

†Cosytec, email : abderrahmane.aggoun@cosytec.com

## Table des matières

# 1 Introduction

L'élimination des symétries dans le cas des problèmes CSP est actuellement un sujet très vivant. En effet la programmation par contraintes, qui permet de résoudre des problèmes de grande complexité combinatoire, nécessite souvent le parcours d'arbres de recherche imposants. Si les capacités purement calculatoires des solveurs sont de plus en plus grandes, tout comme la puissance des nouveaux processeurs, nous continuons à buter sur des problèmes d'une telle complexité qu'il semble impossible de les résoudre en temps raisonnable et en parcourant la totalité de l'arbre de recherche. De nouvelles techniques de résolution ont donc été élaborées ces dernières années afin de réduire cet espace de recherche en tenant compte des propriétés de symétries dans les problèmes. Cependant un certain nombre de questions reste posé quant aux conséquences et à l'efficacité de chacun de ces algorithmes. En effet certains problèmes ont une telle quantité de symétries (voire le problème du golfeur sociable) qu'il peut être contre-productif de vouloir les casser toutes systématiquement. L'objectif de nos travaux, dans le cadre du projet OADymPPaC, est développé des outils de visualisation nous permettant de détecter des symétries pertinentes et d'aider les utilisateurs de telles techniques à améliorer leurs méthodes.

Dans la section 2 de ce rapport nous abordons l'étude des groupes de permutations, très importante dans le cas des symétries dans les CSP, de leurs sous-groupes et des interactions entre ces sous-groupes lorsqu'on élimine une ou plusieurs permutations. En effet si on élimine une symétrie il se peut que d'autres symétries soient également éliminées, partiellement ou complètement. On introduira pour cela la notion de *fonctions préservantes* qui découpent un ensemble d'éléments en sous-ensembles vérifiant une propriété donnée (la symétrie en sera un cas particulier). Dans la section 3, nous donnerons une description plus précise des symétries à travers des exemples et des définitions formelles. On fera également un état de l'art rapide des différentes techniques d'élimination des symétries les plus utilisées et on proposera une implantation en GNU Prolog de l'une d'elle, la SBDS. Enfin dans la section 4 on proposera une heuristique visuelle pour la détection des symétries.

# 2 Fonctions préservantes, groupes et super-groupes

## 2.1 Introduction

Cette partie pose les fondements théoriques de l'utilisation de vues pour détecter des relations entre éléments d'un ensemble, quand les relations sont des groupes de bijections. On suppose que l'on a une collection d'objets qui sont liés par des relations dont on ne connaît pas a priori la nature. Ces relations sont potentiellement révélées par des vues appropriées. Quand une partie des relations possibles est connue, on suppose que l'on est en mesure de les "casser", c'est à dire de produire de nouvelles collections d'objets dans les relations desquels les relations "cassées" ont disparu (c'est à dire que l'on a supprimé les objets "redondants"). On se pose alors le problème de savoir ce que l'on est susceptible d'observer dans les vues qui peuvent être construites avec les objets restants.

Le but est d'appliquer ce travail à la détection de symétries dans les CSP à partir de techniques de visualisation.

Dans les CSP les symétries sont en général définies comme des isomorphismes préservant les solutions. Cette notion s'étend aux "nogoods" (les configurations qui ne sont pas des solutions). Nous étudions donc dans une première partie (section 2.2) la notion générale de "fonction préservante", c'est à dire de fonctions qui sont des isomorphismes préservant une propreté donnée. Il est connu qu'elles ont une structure de groupe et plus particulièrement elles se ramènent au groupe des permutations dont on rappelle quelques propriétés et notations dans la section 2.3. Dans la section 2.4, nous introduisons les notions de "patchwork" et "coproduit" afin de capturer le concept des relations restantes après avoir "cassé" des groupes de fonctions préservantes et on évoque différentes applications possibles". Dans la section 2.5 enfin, on applique ces concepts à la recherche de propriétés de collections d'objets supposés en partie structurés par des relations préservantes connues en partie seulement. On suppose pour ce faire que l'on dispose de méthodes susceptibles de mettre en évidence des groupes de fonctions préservantes. On est évidemment intéressé à l'utilisation de techniques de visualisation, mais cet aspect ne sera effectivement mis en œuvre que dans les chapitres suivants.

## 2.2 Fonctions préservantes

**Definition 2.1** [Fonction préservante de $P$ sur $E$]

Etant donnée une propriété $P$ et un ensemble $E$, une *fonction préservante de $P$ sur $E$*, $f$, est une bijection sur $E$ telle que

$\forall x \textit{ dans } E,\ P(x) \Rightarrow P(f(x)) \textit{ et } P(x) \Rightarrow P(f^{-1}(x))$

On dira qu'une fonction est préservante sans plus de précisions si elle est préservante quelques soient $P$ et $E$. On notera $\circ$ la composition de fonctions.

**Proposition 2.1** [Quelques propriétés]

1. La fonction identité, dénotée $i$, est une fonction préservante. $i$ vérifie trivialement : $i^2 = i$ (idempotence) et donc $i^{-1} = i$, et pour toute fonction $f : i \circ f = f \circ i = f$.
2. Si $f$ est une fonction préservante de $P$ sur $E$, $f^{-1}$ est une fonction préservante de $P$ sur $E$.
3. Si $f$ et $g$ sont des fonctions préservantes de $P$ sur $E$, $f \circ g$ est est une fonction préservante de $P$ sur $E$.
4. Si $f$ est une fonction préservante de $P$ sur $E$ et tous les éléments de $E$ satisfont $P$ ou $P'$ exclusivement, alors $f$ est une fonction préservante de $P'$ sur $E$.
5. Si $f$ est une fonction préservante de $P$ sur $E$ et s'il existe un entier $n$ positif tel que $f^{n+1} = i$, alors $f$ est fonction génératrice d'un groupe de fonctions préservantes de $P$ sur $E$ de la forme $\{i, f, f^2, \ldots, f^n\}$. Un tel entier $n$ existe toujours si le sous-ensemble de $E$ qui verifie $P$ est fini.

**Example 2.1** Soit une feuille transparente sur laquelle est dessiné un carré. La fonction de rotation selon un axe perpendiculaire à la feuille et passant par le centre du carré est une fonction préservante de la figure observée sur l'ensemble des positions que peut prendre la feuille dans l'espace.

1. La fonction identité correspond à l'absence de mouvement.
2. Si $r$ est une rotation de 90 degrés, $r^{-1}$ est la rotation inverse ($-90^o$).
3. $r^2$ et $r^3$ préservent la figure observée.

4. Si au lieu d'un carré on trace une patate irrégulière, le figure obtenue par $r$ reste une patate irrégulière (pas forcément la même, la propriété invariante est le fait d'être une patate irrégulière).
5. $r$ vérifie $r^4 = i$ et est donc fonction génératrice du groupe $\{i, r, r^2, r^3\}$ de taille 4.

PROOF. [Preuve de la propriété 2.1]

1. Immédiat.
2. Par définition.
3. Si $f$ et $g$ sont des fonctions préservantes de $P$ sur $E$, $f \circ g$ vérifie
$\forall x \; dans \; E, \;\; P(x) \Rightarrow P(f(x))$ et donc $P(g(f(x)))$
puisque la même propriété s'applique à $g$. De même pour $(f \circ g)^{-1} = (g^{-1} \circ f^{-1})$.
4. Si $f$ vérifie, $\forall x \; dans \; E, \;\; P(x) \Rightarrow P(f(x)) \; et \; P(x) \Rightarrow P(f^{-1}(x))$, alors $f$ vérifie $\forall x \; dans \; E, \;\; P'(x) \Rightarrow P'(f(x)) \; et \; P'(x) \Rightarrow P'(f^{-1}(x))$. En effet $P'(x)$ correspond au sous-ensemble des éléments de $E$ qui vérifient $P'$. Comme $f$ est une bijection sur $E$ et que tout élément de $E$ vérifie soit $P$ soit $P'$, l'application de $f$ à un élément de l'un des sous-ensembles ne peut que donner un élément du même sous-ensemble, sinon l'une des conditions sur $P$ serait violée.
5. Il suffit de constater que comme les compositions successives de $f$ sont des fonctions préservantes de la même propriété, s'il existe un entier positif $n$ fini tel que $f^{n+1} = i$, l'ensemble des fonctions $\{i, f, f^2, \ldots, f^n\}$ forme bien un groupe ($i$ neutre pour la composition, associativité de la composition, chaque élement $g$ a un inverse : si $g = f^j$, alors $g^{-1} = f^{n+1-j}$ et $g \circ g^{-1} = g^{-1} \circ g = i$). Comme $f$ est une bijection, si le sous-ensemble de $E$ des éléments vérifiant $P$ est fini, il existe necessairement un entier positif $n$ tel que $f^{n+1} = i$ (on retombe sur le premier élément du groupe généré), puisqu'un élément de $E$ ne peut être le résultat que de l'application de $f$ à un seul autre élement de $E$ (donc nécessairement élément initial, dénoté $i$, du groupe).

□

## 2.3 Groupe des fonctions préservantes et représentation

Une fonction préservante est une bijection sur le sous-ensemble des éléments de $E$ qui verifient $P$. Par abus de notation, et s'il n'y a pas d'ambiguité, on confondra dorénavant le nom de la propriété et l'ensemble de éléments de $E$ qui vérifient $P$. Toute bijection dans $E$ peut être représentée comme une permutation des éléments de $E$ qui à chaque élément de $E$ fait correspondre un élément de $E$. Si la bijection est une fonction préservante de $P$, elle correspond dans $P$ à une permutation des éléments de $P$.

Il s'en suit que si $P$ est fini de cardinalité $N$, l'ensemble des fonctions préservantes de $P$ sur $E$ forme un groupe de cardinalité $N!$. Ceci résulte du fait que chaque fonction préservante est génératrice d'un sous-groupe, que deux fonction préservante sont génératrices d'un sous-groupe éventuellement bien supérieur à l'union des fonctions des deux groupes, etc... Mais tous les sous-groupes obtenus sont des sous-groupes du groupe des permutations de $P$ que nous appellerons $G_P$.

Il ressort de ces observations deux faits principaux : premièrement l'étude des fonction préservantes, groupes et sous-groupes relève essentiellement de l'étude du groupe des permutations auquel elles se ramènent. Ceci va nous conduire à adopter certaines notations pour décrire les fonctions préservantes.

Par ailleurs la recherche de fonctions préservantes d'une propriété donnée $P$ peut paraître triviale puisque toute permutation des éléments de $P$ est une fonction préservante. Ceci est vrai si l'ensemble $P$ est connu. Or dans le problème auquel nous nous intéressons de telles fonctions préservantes sont supposées connues à l'énnoncé du problème ou découvertes au fur et à mesure de la construction de $E$. On verra qu'en pratique cette situation amène à ne s'intéresser qu'à des sous-groupes remarquables de $G_P$.

### 2.3.1 Groupe des permutations : représentation

Afin de pouvoir présenter des exemples, nous adopterons la notation usuelle des groupes illustrée par quelques exemples.

L'ensemble $P$ (ensemble des éléments vérifiant une propriété $P$) de cardinalité $N$ est dénoté par l'ensemble des entiers $\{1, \ldots, N\}$. Une bijection sera donc représentée par une liste de couples indiquant les éléments en correspondance. Ainsi une bijection restreinte à la permutation $p$ de deux éléments $i$ et $j$ sera représentée par les couples $[(i, j), (j, i)]$, les autres éléments étant inchangés. Si $N = 4$ et $p$ est la permutation des éléments 2 et 3 ($p = [(2, 3), (3, 2)]$), la fonction $p$ fait passer de l'état $\{1, 2, 3, 4\}$ à l'état $\{1, 3, 2, 4\}$.

**Example 2.2** [cycle de 4 éléments]

La figure 1 illustre un cycle de 4 éléments défini par $c = [(1, 2), (2, 3), (3, 4), (4, 1)]$.

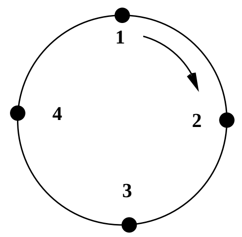


FIG. 1 – Groupe d'un cycle de 4 éléments

$c$ est générateur d'un groupe de taille 4 isomorphe au groupe des rotations de l'exemple 2.1 : $\{i, c, c^2, c^3\}$.

On trouvera dans l'annexe 1 des exemples de groupes avec leur description.

### 2.3.2 Groupe des permutations et sous-groupes

Les groupes et plus particulièrement le groupe (fini) des permutations ont fait l'objet de nombreuses études et réalisations. Le système GAP en particulier permet l'études d'un groupe donné, de ses fonctions génératrices, de sa taille, etc.

Nous ne rapellerons ici que quelques éléments utiles à notre étude.

Comme nous l'avons vu l'ensemble des fonctions préservantes d'une propriété $P$ donnée forme un groupe $G_P$ de cardinalité $N!$. En pratique on s'intéressera à des sous-groupes de $G_P$, voire à des sous-groupes de sous-groupes distingués de $G_P$.

**Example 2.3** [Le groupe $Geom$]

On considère le groupe des transformations géométriques usuelles

$Geom = \{i, m1, m2, s1, s2, r, o, r1\}$

symétries par rapport aux médiannes horizontale ($m1$) et verticale ($m2$), par rapport aux diagonales droite ($s1$) et gauche ($s2$), et rotations respectivement de 90 ($r$), 180 ($o$) et 270 ($r1$) degrés positifs).

On indique quelques données utiles pour ce groupe de taille 8. L'ensemble considéré est l'ensemble des permutations de 4 éléments numérotés de 1 à 4. Le groupe de permutation obtenu est isomorphe aux transformations géométriques d'un carré. Pour s'en rendre compte il suffit de numéroter chaque coin du carré de 1 à 4 (sens trigonométrique en partant du coin supérieur gauche).

```
Operators: i m1 s1 r1 r m2 s2 o
i:   []
m1:  [(1,3),(2,4),(3,1),(4,2)]
s1:  [(1,4),(4,1)]
r1:  [(1,3),(2,1),(3,4),(4,2)]
r:   [(1,2),(2,4),(3,1),(4,3)]
m2:  [(1,2),(2,1),(3,4),(4,3)]
s2:  [(2,3),(3,2)]
o:   [(1,4),(2,3),(3,2),(4,1)]

Idempotent operators (op.op=op allbut identity i): none
Self inverse operators (op^2=i allbut identity): m1 s1 m2 s2 o
Inverse pairs (allbut identity and self inverse): [r1,r]
Commutative pairs (allbut previous ops):
[m1,m2] [m1,o] [s1,s2]  [s1,o] [r1,o] [r,o] [m2,o] [s2,o]

Simplified composition  table:
---------------
m1 . s1 = r1     m2 . s1 = r      s1 . m1 = r      s2 . m1 = r1
m1 . r1 = s1     m2 . r1 = s2     s1 . r1 = m2     s2 . r1 = m1
m1 . r  = s2     m2 . r  = s1     s1 . r  = m1     s2 . r  = m2
m1 . m2 = o      m2 . s2 = r1     s1 . m2 = r1     s2 . m2 = r
m1 . s2 = r      m2 . o  = m1     s1 . s2 = o      s2 . o  = s1
m1 . o  = m2                      s1 . o  = s2

r . m1 = s1      r1 . m1 = s2
r . s1 = m2      r1 . s1 = m1
r . r  = o       r1 . r1 = o
r . m2 = s2      r1 . m2 = s1
r . s2 = m1      r1 . s2 = m2
r . o  = r1      r1 . o  = r
--------------
```

## 2.4 Patchworks et super-groupes

Tout sous-groupe $G$ de $G_P$ induit une partition de $P$ en considérant que tous les les éléments qui se correspondent par une fonction de $G$ sont dans une même classe.

Ceci est formalisé par la notion d'équivalence induite par un (sous-)groupe.

**Definition 2.2** [Equivalence et partition induite par un groupe]
Etant donné un ensemble $E$ et un groupe $G$ de fonctions préservantes de la propriété $P$, défini par l'ensemble de ses fonctions, la relation d'équivalence induite par $G$ sur l'ensemble $P$ est définie par $x \equiv y\ si\ \exists f, f \in G, y = f(x)$. On notera $\overline{x}$ la classe d'équivalence dans laquelle se trouve l'élément $x$, sachant que n'importe quel élément de la classe peut être utilisé.

Notons que pour un groupe donné tout élément de $P$ appartient à une seule classe d'équivalence. Chaque sous-groupe de $G_P$ induit une partition. Par ailleurs rien ne justifie a priori de choisir un élément plutôt qu'un autre comme représentant de la classe.

**Example 2.4** Reprenons l'exemple 2.1. Quelque soit la position initiale du carré par rapport à l'observateur, toutes les figures obtenues par application d'une fonction du groupe des rotations sont rigoureusement identiques. Elles sont dans la même classe d'équivalence. Le groupe des rotations induit donc une partition sur l'ensemble des positions possibles de la feuille de papier dans l'espace.

Le problème principal auquel nous nous intéressons maintenant est de savoir quelles relations sont susceptibles de lier les classes d'équivalence entre elles ou, plus précisemment, dans le cas où l'on ne conserve qu'un représentant de chaque classe, comment sont-ils reliés entre eux. Pour ce faire nous introduisons la notion de patchwork et une nouvelle relation de composition d'ensembles de fonctions, dite coproduit.

**Definition 2.3** [Patchwork et coproduit]
On appelle *patchwork* et on notera $\overline{F} = \{f1; f2; \ldots; f_m\}$ un ensemble de fonctions préservantes (l'ordre de éléments est donc non signifiant). L'ensemble des fonctions utilisées dans le patchwork $\overline{F}$ sera dénoté $F = \{f1, f2, \ldots, f_m\}$.
Les opérations ensemblistes (union, inclusion, ...) seront utilisées sans plus de précision s'il n'y a pas de confusion possible sur les patchworks ou les ensembles.
On appelle *coproduit* et on notera $\star$ la composition de deux patchworks, défini comme l'ensemble de toutes les compositions d'une fonction du premier avec une fonction du second, dans cet ordre.

La raison pour utiliser un point-virgule au lieu d'une virgule pour le patchwork est l'interprètation du point-virgule qui peut être vu comme une disjonction. La composition pour un patchwork est l'opération $\star$. Elle n'est pas définie sur les ensembles.

**Example 2.5** [Coproduit]
Soit le groupe géométrique $Geom = \{i, m_1, m_2, s_1, s_2, r, o, r1\}$,
$\{i; m_1\} \star \{o; m_2\} = \{i \circ o;\ i \circ m_2;\ m_1 \circ o;\ m_1 \circ m_2\} = \{o;\ m_2\}$

**Proposition 2.2** [caractérisation d'un groupe par coproduit]
Un ensemble $G$ de fonctions bijectives sur $E$ est un groupe si et seulement si $\overline{G}^2 = \overline{G}$.

Proof. Si $G$ est un groupe, il est de la forme $G = F \cup \{i\}$ et donc $\overline{G}^2 = \overline{F}^2 \cup \overline{F} \cup \{i\} = \overline{F}^2 \cup \overline{G}$. Il en résulte que $\overline{G} \subseteq \overline{G}^2$. Par ailleurs on a par définition d'un groupe $\overline{G}^2 \subseteq \overline{G}$ puisque $\overline{G}^2$ est la composition deux à deux de toutes les opérations de $G$ dont le résultat est un élément de $G$.

On montre que si le patchwork $\overline{G}$ est idempotant alors $G$ est un groupe. Par hypothèse : $\overline{G}^2 = \overline{G}$. On montre en trois étapes que sous cette hypothèse, (1) toutes les compositions d'opérations de $G$ ont un ésultat dans $G$, (2) toute fonction de $G$ à un inverse dans $G$, et (3) l'élément $i$ appartient à $G$.

(1) est évident du fait de la définition du coproduit.

(2) résulte de l'observation que la composition d'une fonction $f$ avec les $n$ fonctions de $G$ (si le cardinal de $G$ est $n$) définit $n+1$ fonctions distinctes qui transforment le point d'origine en $n+1$ points différents puisque toutes les fonctions composées sont des bijections. Or de part l'hypothèse seules $n$ fonctions existent. Donc l'une des $n$ compositions a pour résultat le point d'origine. On en déduit que pour toute fonction $f$ de $G$ différente de l'identité, il existe une fonction $f'$ de $G$ différente de l'identité, telle que $f \circ f' = i$. Le même raisonnement s'applique à $f'$ pour laquelle il existe une fonction $f''$ telle que $f' \circ f'' = i$. En appliquant $f''$ à droite de la première égalité : $f \circ f' \circ f'' = i \circ f'' = f''$, on obtient $f = f''$ et donc $f$ et $f'$ verifient : $f \circ f' = f' \circ f = i$. Donc tout élément de $G$ a un inverse dans $G$.

(3) il en résulte également que $i$ est dans $G$ puisque c'est le résultat de l'application d'une fonction de $G$ à son inverse également dans $G$, et donc, selon l'hypothèse, ce résultat appartient à $G$.

□

On s'interesse maintenant plus particulièrement à comprendre le comportement de groupes quand ils sont considérés comme une seule entité, c'est à dire aux classes d'équivalences engendrées par les groupes de fonctions préservantes. On considère donc une nouvelle famille de fonctions préservantes : celles qui préservent un groupe donné. D'où la notion de patchwork associé à un groupe. Un groupe $G$ peut être considéré comme un patchwork $\overline{G}$ constitué de l'ensemble intégral de ses fonctions (et non de quelques générateurs), incluant l'identité.

**Definition 2.4** [Patchwork préservant associé à un groupe]

Etant donné un sous-groupe $G$ de $G_P$ et un ensemble $F$ de fonctions préservantes de $P$ distinctes des fonctions de $G$, le patchwork $\overline{F}$ est dit *patchwork préservant du groupe* $G$, si l'image de $G$ par chaque fonction de $\overline{F}$ est la même et que tous les éléments de l'image appartiennent à une même classe d'équivalence.

Plus formellement : soit $\overline{G_f}$ l'image de $\overline{G}$ par une fonction $f$ de $\overline{F}$,

1. $\forall f \in \overline{F},\ \forall h, h' \in \overline{G_f}\ \exists g \in \overline{G}$ tel que $h = g \circ h'$.
2. $\forall f, f' \in \overline{F},\ \forall h \in \overline{G_f},\ \forall h' \in \overline{G'_f}\ \exists g \in \overline{G}$ tel que $h = g \circ h'$.

On notera $\overline{F}^{-1}$ et appellera *patchwork inverse* de $\overline{F} = \{f1; f2; \ldots; f_m\}$ le patchwork $\overline{F}^{-1} = \{f1^{-1}; f2^{-1}; \ldots; f_m^{-1}\}$.

La première condition exprime que le groupe $G$ est préservé lorsque l'on applique chacune des fonctions de $\overline{F}$ individuellement, et la seconde que chaque fonction de $\overline{F}$ produit la même classe d'équivalence. Par la première condition chaque fonction de $\overline{F}$ produit bien une classe d'équivalence et la deuxième exprime que les éléments de deux classes d'équivalences obtenues par deux fonctions de $\overline{F}$ sont liés par une fonction de $\overline{G}$ et donc appartiennent au même groupe, donc à la même classe d'équivalence. On justifiera plus bas la notion de patchwork inverse.

On peut donner d'autres caractérisations éventuellement utiles pour élaborer des algorithmes de recherche de fonctions préservantes.

**Proposition 2.3** [Autres caractérisations]

Etant donné un sous-groupe $G$ de $G_P$ et un ensemble $F$ de fonctions préservantes de $P$ distinctes des fonctions de $G$, le patchwork $\overline{F}$ est *patchwork préservant du groupe* $G$ si deux des trois conditions suivantes sont satisfaites :

$\forall f \in \overline{F}$

1. $\forall g \in \overline{G}\ \exists g' \in \overline{G}\ t.q.\ g \circ f = f \circ g'$ (préservation)
2. $\forall f' \in \overline{F}\ \exists g \in \overline{G}\ t.q.\ f \circ g = f'$ (reconstruction)
3. $\forall g \in \overline{G},\ \forall f' \in \overline{F}\ \exists g' \in \overline{G}\ t.q.\ \ g \circ f = f' \circ g'$ (agrégation)

Les conditions sont illustrées sur la figure 2.

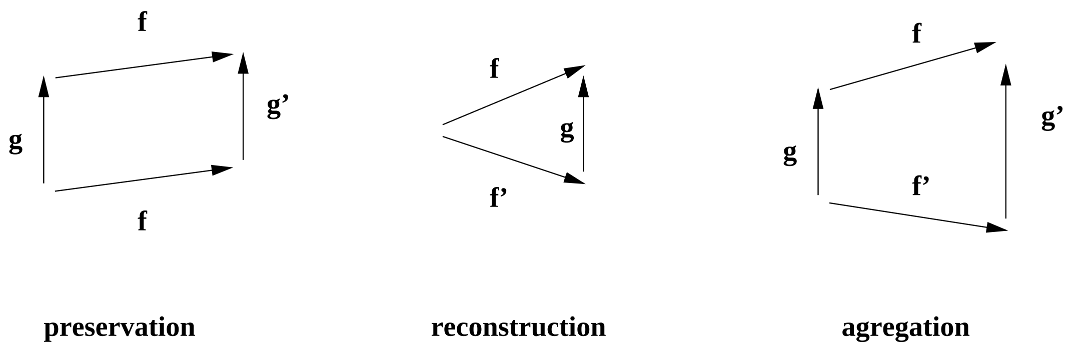


Fig. 2 – Illustration des propriétés (liées à chaque condition) d'un patchwork préservant

La première condition, dite de préservation, stipule que les deux éléments, images par une fonction de $\overline{F}$ de deux éléments de $\overline{G}$, doivent être liés par une relation de $\overline{G}$ (éventuellement différente). Autrement dit les images de deux éléments de la même classe d'équivalence par chaque fonction de $\overline{F}$ sont dans la même classe d'équivalence.

La seconde condition, dite de reconstruction, stipule que deux images par deux fonctions de $\overline{F}$ d'un même élément (supposé de $\overline{G}$) sont liés par une relation de $\overline{G}$ et donc dans la même classe déquivalence. Elle caractérise le fait qu'à partir d'un élément quelconque on peut reconstruire une classe d'équivalence complète.

Enfin la troisième condition, dite d'agrégation, stipule les images de deux points d'une même classe d'équivalence par deux fonctions distinctes de $\overline{F}$ sont dans la même classe d'équivalence (on ne peut pas élargir une classe déquivalence avec un patchwork préservant).

Proof. (à faire)

Noter que les conditions de la définition 2.4 sont des formulations des conditions 1 et 3 de la proposition.

On montre que deux conditions suffisent pour assurer que l'image par $\overline{F}$ de $\overline{G}$ (obtenue par le coproduit de $\overline{G}$ et $\overline{F}$) est un groupe avec les fonctions de $\overline{G}$, car avec les conditions et par transitivité le produit de deux fonctions du groupe est une fonction du groupe), tous les éléments de l'image de $\overline{G}$ par $\overline{F}$ sont liés par une relation de $\overline{G}$.

□

**Example 2.6** [Patchwork préservant]

Le patchwork $\overline{F} = \{o; m_2\}$ de l'exemple 2.5 préserve le groupe $\overline{G} = \{i; m_1\}$ (c'est bien un groupe du fait que $m_1^2 = i$). En effet l'image (unique) par $\overline{F}$ de $\overline{G}$ est $\{o; m_2\}$ qui verifie $o \circ m_1 = m_2$ et $m_2 \circ m_1 = o$.

Nous avons développé un programme pour obtenir tous les patchworks possibles des sous-groupes d'un groupe donné.

Pour le groupe des transformations géométriques $Geom$, les groupes préservés $g\tilde{}$ et les patchworks préservants associés $f\tilde{}$ sont les suivants : (un patchwork $\overline{F}$ est noté `f~`)

```
g~: [i,m1]   f~: [m2,o]  f~^2 = g~
g~: [i,m1,m2,o]   f~: [r,r1,s1,s2]  f~^2 = g~
g~: [i,m2]   f~: [m1,o]  f~^2 = g~
g~: [i,o]   f~: [m1,m2]  f~^2 = g~
g~: [i,o]   f~: [r,r1]  f~^2 = g~
g~: [i,o]   f~: [s1,s2]  f~^2 = g~
g~: [i,o,r,r1]   f~: [m1,m2,s1,s2]  f~^2 = g~
g~: [i,o,s1,s2]   f~: [m1,m2,r,r1]  f~^2 = g~
g~: [i,s1]   f~: [o,s2]  f~^2 = g~
g~: [i,s2]   f~: [o,s1]  f~^2 = g~
```

**Proposition 2.4** [Propriétés des patchworks préservants et super-groupe]

Etant donné un sous-groupe $G$ de $G_P$ et $\overline{F}$ un patchwork dont toutes les fonctions sont différentes de celles de $G$

1. si $\overline{F}$ est un patchwork préservant de $G$, alors $|\overline{F}| = |\overline{G}|$ (même cardinalité).
2. Si $\overline{F}$ est un patchwork préservant de $G$ alors
   $\overline{G} \star \overline{F} = \overline{F} \star \overline{G} = \overline{F}$
3. Si $\overline{F}$ est un patchwork péservant de $\overline{G}$ alors
   $\overline{F} \star \overline{F}^{-1} = \overline{F}^{-1} \star \overline{F} = \overline{G}$
4. Si $\overline{F}$ est un patchwork préservant de $G$, alors $\overline{F}^2 = \overline{G}$ est une condition nécessaire et suffisante pour que l'union des fonctions de $\overline{F}$ et $\overline{G}$ forme un groupe. On dira dans ce cas que le $\overline{F}$ est un patchwork péservant *total* de $G$ par rapport à $G \cup F$.
5. Si $\overline{F}$ est un patchwork préservant de $G$, $\overline{F}$ est générateur d'un groupe (de groupe) appelé *super-groupe*. Pour ce super-groupe, $\overline{G}$ est l'identité et le patchwork préservant inverse de $\overline{F}$ est $\overline{F}^{-1}$.

Ainsi par exemple, si on prend le groupe des rotations de $Geom$ (cf exemple 2.6) $G = \{i,\ r,\ o,\ r1\}$ le patchwork $\overline{F} = \{m_1;\ m_2;\ s_1;\ s_2\}$ est préservant du groupe $G$. De plus on observe que l'union des fonctions de $G$ et $F$ est le groupe $Geom = \{i,\ m_1,\ m_2,\ s_1,\ s_2,\ r,\ o,\ r1\}$, donc, d'après la propriété 3, il vérifie aussi $\overline{F}^2 = \overline{G}$ ; et donc $G \cup F$ est un super-groupe de taille 2.

En supposant qu'un groupe $G$ puisse être représenté par une figure rassemblant ses éléments en une seule figure dont les propriétés visuelles sont celles des opérations du groupe (par exemple : invariance par application des relations du groupe aux éléments de la figure) un patchwork $\overline{F}$ préservant du groupe $G$ peut s'interprèter de trois manières :

- Conservation des proprétés des figures transformées par chacune des fonctions du patchwork.
- Dans le cas où seuls quelques éléments de la figure intiale sont considérés, reconstitution d'une figure intégrale (transformée) satisfaisant les propriétés du groupe original.
- Dans deux figures (même partielles) transformées avec des fonctions du patchwork préservant, deux éléments pris dans chaque figure transformée sont liés par une relation du groupe initial.

**Example 2.7** [Applications du patchwork]

Considérons le patchwork $\{m_2;\ o\}$ préservant du groupe $\{i,\ m_1\}$ des transformations géométriques (exemple 2.6). La figure 3 illustre les différentes interprètations possibles sur une figure simple invariante par symétrie selon la médiane horizontale (fonction $m_1$).

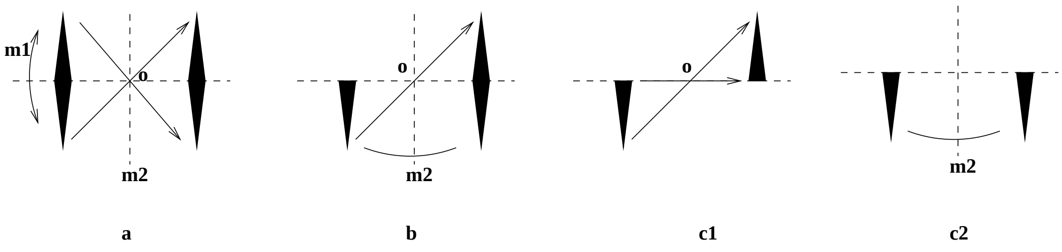


FIG. 3 – Effet du patchwork $\{m_2;\ o\}$ sur le groupe médiane horizontale

- La figure **a** montre la conservation de la figure de gauche (aiguille de boussole) par la transformation représentée par le patchwork. (Noter que la transformation laisse la figure invariante)
- La figure **b** illustre la reconstitution de la figure en apliquant le patchwork sur la figure partielle (mais reconstituable par application du groupe).
- Enfin les figures **c1** et **c2** montrent deux figures partielles (transformées de la même figure partielle originale), et dont les éléments sont en correspondance par l'une des fonctions du groupe (ici $m_1$).

L'exemple suivant montre que si une figure n'est pas forcement préservée par un patchwork, ses propriétés d'invariance pour le groupe original, elles, le sont.

**Example 2.8** [Applications du patchwork : préservation du groupe et non de la figure]

La figure 4-**a** est une figure invariante par le groupe des rotations de *Geom* : $G = \{i,\ r,\ o,\ r1\}$.

Sa transformation par le patchwork $\overline{F} = \{m_1;\ m_2;\ s_1;\ s_2\}$ donne une figure analogue (figure 4-**b**) mais différente que l'on obtient par application de l'une des fonctions de $F$. Cette figure est également invariante par le groupe des rotations.

Si maintenant on suppose qu'on ne dispose que d'une collection de fragments élémentaires de la figure, il est possible de reconstituer la figure transformée à partir

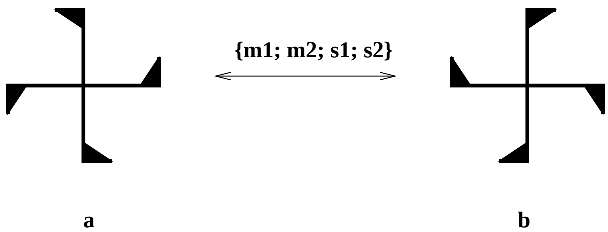


FIG. 4 – Effet du patchwork $\{m_1;\ m_2;\ s_1;\ s_2\}$ sur une figure invariante par rotation seulement

d'un seul des fragments en appliquant le patwork à ce fragment. Ceci est illustré par la figure 5-**a**.

Si par contre on dispose de fragements de la figure originale et de sa transformée, la figure resonstituée (figure 5-**b**) est la superposition des deux figures et elle a les propriétés du groupe complet *Geom*.

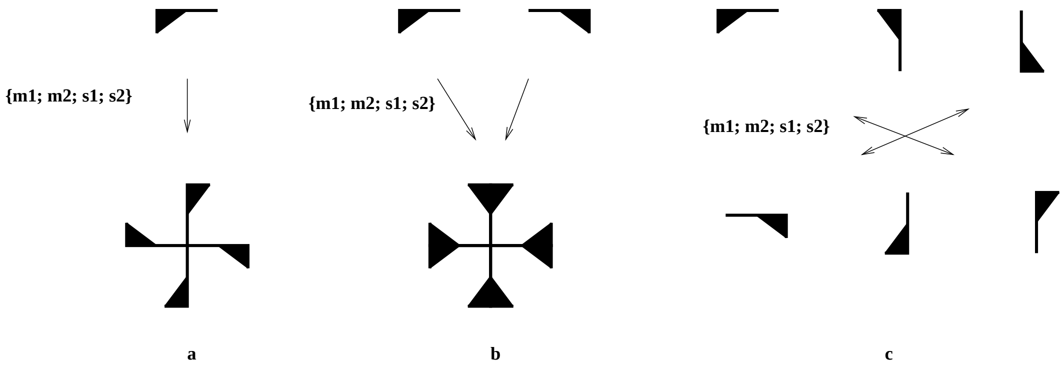


FIG. 5 – Effet du patchwork $\{m_1;\ m_2;\ s_1;\ s_2\}$ sur des fragments de la figure construite avec le groupe $\{i,\ r,\ o,\ r1\}$

Enfin si l'on dispose de fragments éparses des deux figures (originale et transformée (figure 5-**c**) on observe que chaque couple de fragment est lié par une des relations de $F$.

PROOF. [Preuve de la propriété 2.4] (à faire)

□

## 2.5 Application à la recherche de propriétés de collections d'objets

Nous sommes en mesure d'expliquer maintenant la manière d'utiliser cette méthode pour détecter des propriétés de collections d'objets à l'aide de fonctions préservantes de propriétés.

On dispose au départ d'une collection d'objets qui tous dérivent les uns des autres par l'application d'un ensemble de transformations a priori inconnu. La figure 6 montre une collection de 8 objets qui tous dérivent d'un seul des objets par application des 7 transformations du groupe $Geom = \{i, m1, m2, s1, s2, r, o, r1\}$

(symétries par rapport aux médiannes horizontale et verticale, par rapport aux diagonales droite et gauche, et rotations respectivement de 90, 180 et 270 degrés positifs). A côté de cette figure sont données deux représentations synthétiques de ces objets susceptibles de mettre en évidence tout ou partie de ces relations. On les appelle des *vues*. La **vue a** rassemble tous les objets en une seule image qui est invariante par chacune des transformations du groupe *Geom*. La **vue b** a deux composants qui rassemblent respectivement les objets plutôt horizontaux et les objets plutôt verticaux. Elles sont manifestement invariantes par les transformations du sous-groupe $\{i, m1, m2, o\}$. Dans les deux cas, on suppose que l'on ne connaît rien des relations qui lient les objets. Ces deux vues servent à les révéler. La première vue (**vue a**) met clairement en évidence toutes les relations du groupe *Geom*. La seconde vue proposée (**vue b**) ne met en évidence que les relations $\{i, m1, m2, o\}$.

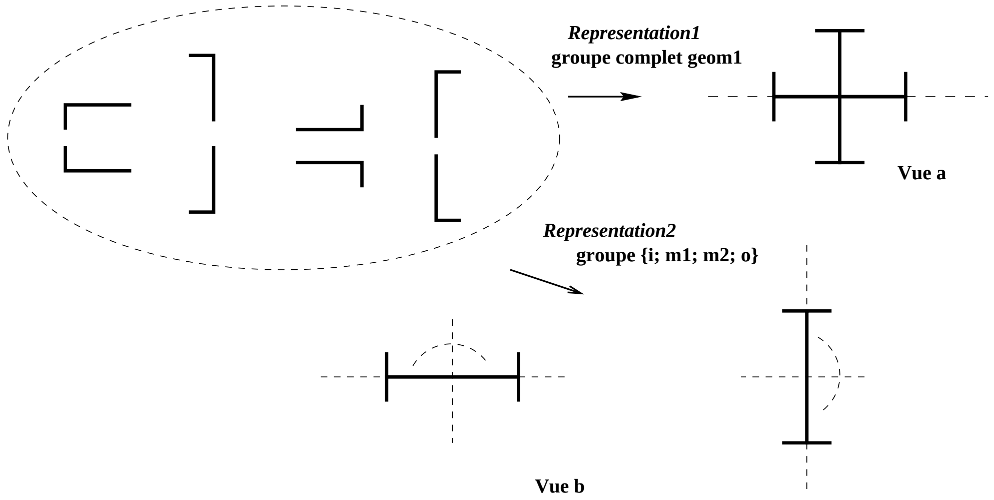


FIG. 6 – Application : mise en évidence de groupes totaux (Vue$_a$) ou partiels (Vue$_b$)

Le problème auquel on s'intéresse ici est ce qui se passe lorsque, sur les vues proposées, on ne distingue qu'une partie des relations et non toutes les relations possibles. La **vue b** par exemple ne permet de détecter que les relations $\{i, m1, m2, o\}$. Chaque vue, on l'aura compris, est fabriquée en rassemblant des objets entre eux de manière à former une "vue synthétique". On va donc examiner différents cas où, en supposant que l'on ait repéré un sous-groupe de relations liant les objets à partir des **vues a** ou **b**, on regarde les différentes vues que l'on peut construire à partir des seuls objets représentants quelconques des classes d'équivalence obtenues. Les fonctions préservantes des classes d'objets représentent les relations possibles entre ces représentants. La question que l'on se pose est si les relations qui n'ont pu être repérés sur les vues initiales (avec tous les objets) peuvent l'être encore sur les nouvelles vues contruites avec les représentants des classes seulement.

La figure 7 illustre un premier cas où, à partir de la **vue a**, seul un sous-groupe $\{i, m1, m2, o\}$ a été reconnu (on aurait pu tout aussi bien reconnaître le sous-groupe $\{i, r, o, r1\}$).

Deux classes d'équivalence ont pu être formées et il y a plusieurs manières de choisir un représentant de la classe. On considère ici 4 des 16 possibilités. On désignera le fait de choisir les représentants de chaque classe *casser les relations*. Si donc les

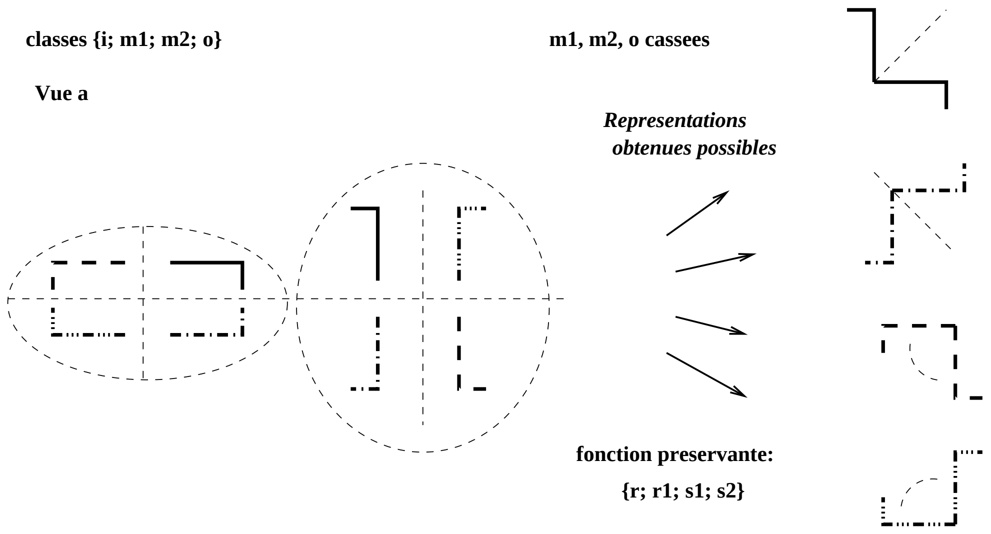


FIG. 7 – Cas 1 : le sous-groupe $\{i, m1, m2, o\}$ a été reconnu sur la **vue a**, effet sur la **vue a** après que les relations aient été cassées

relations du groupe $\{i, m1, m2, o\}$ ont été cassées, il ne reste qu'un couple d'objets. La figure 7 montre quatre des vues possibles produites selon le même principe et qui dépendent des représentants choisis.

On observe que la fonction préservante est $\{r; r1; s1; s2\}$ et que les vues obtenues mettent en évidence clairement une des relations de la fonction préservante, mais une seule à la fois. Il est impossible de savoir a priori quelle relation sera mise en évidence, mais comme la fonction préservante est un patchwork préservant total, on est certain de pouvoir observer au moins l'une des relations.

Le même résultat peut être observé sur la figure 8 où, à partir de la **vue b**, seul un sous-groupe $\{i, m1\}$ a été reconnu (on aurait pu tout aussi bien reconnaître les sous-groupes $\{i, m2\}$ ou $\{i, o\}$).

Quatre classes d'équivalence sont cette fois formés et il y a également 16 manières de choisir les représentants des classses. Deux sont représentées.

Cette fois le groupe concerné par la **vue b** est $\{i, m1, m2, o\}$, la fonction préservante est $\{m2, o\}$ et elle est totale pour ce groupe (elle ne le serait pas si la **vue a** avait été utilisée). Sur la figure 8 seules les deux nouvelles **vues b** sont représentées ; chacune met en évidence cette fois les deux relations du patchwork.

Finalement la figure 9 illustre le cas où le patchwork n'est pas total. Seule la relation $m1$ a été reconnue bien que le groupe sous-jacent à la **vue a** soit le groupe $Geom$.

Là aussi 16 nouvelles **vues a** sont possibles mais aucune ne met clairement en évidence les relations du patchwork préservant $\{m2, o\}$ qui n'est pas total pour $Geom$. Les deux vues présentées contiennent ainsi des relations dans $\{s1, s2, r, r1\}$. Une seule chose est sûre : aucune ne peut mettre en évidence la relation $m1$.

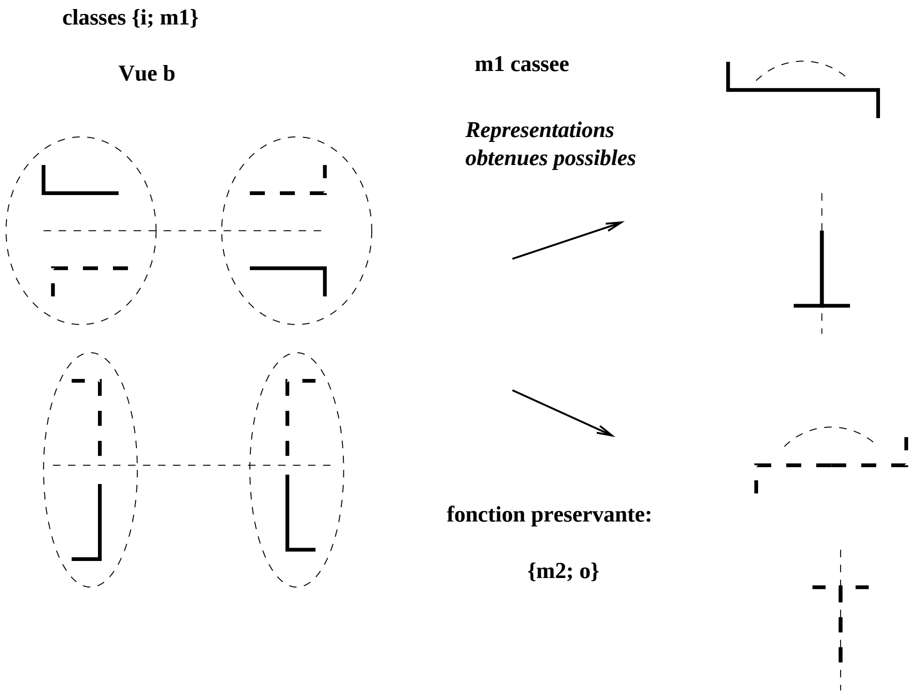


Fig. 8 – Cas 2 : le sous-groupe $\{i, m1\}$ a été reconnu sur la **vue b**, effet sur la **vue b** après que la relation ait été cassée

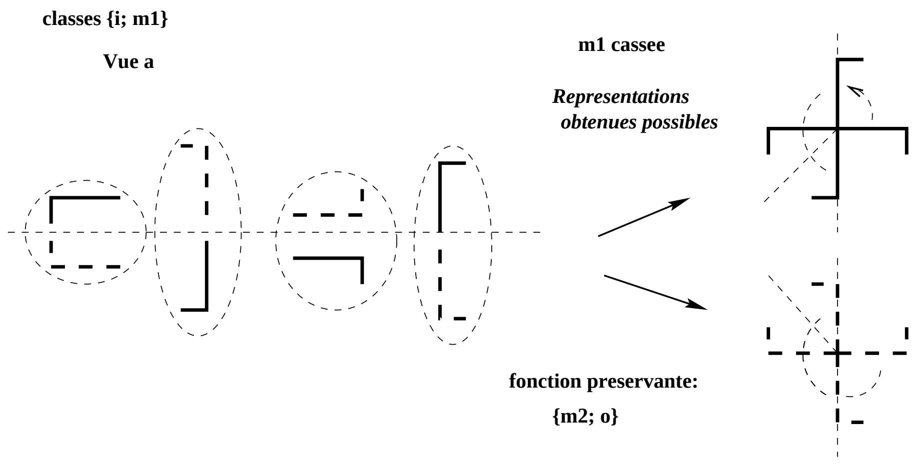


Fig. 9 – Cas 3 : le sous-groupe $\{i, m1\}$ seulement a été reconnu sur la **vue a**, effet sur **vue a** après que la relation ait été cassée

## 2.6 Conclusion

Dans cette section nous avons présenté une étude encore imcomplète des fondements théoriques de l'utilisations de vues pour détecter des relations entre éléments d'un ensemble, quand les relations sont des groupes de bijections. Plusieurs voies restent à explorer :

- La notion de patchwork préservant total reste à approfondir. En effet elle joue un rôle intéressant lorsque le groupe susceptible d'être représenté par une vue est inconnu.
- Cette étude suppose évidemment que l'on soit capable de produire des vues ayant une capacité "révélante" de groupes particuliers. Il se trouve par exemple que les vues var/val utilisées dans les chapitres suivants pour l'analyse des symétries dans les CSP ont cette capacité (ceci est dû à la manière dont sont spécifiés les isomorphismes de configurations).
- L'analyse des groupes et des patchworks associés à des sous-groupes de groupes est faite à l'aide de programmes réalisés en GNU-Prolog. Cette approche permet des analyses de groupes relativement petits (jusqu'a une taille de l'ordre de la centaine de fonctions) avec une implantation particulièrement facile en Prolog mais peu efficace (utilisation massive de `findall/3` et de `sort/2` par exemple). Une efficacité meilleure pourrait être obtenue en introduisant des passerelles avec des systèmes optimisés pour l'analyse de groupes de permutations tels que GAP d'une part ; et d'autre part en utilisant les caractérisations ou les propiétés remarquables (cf propriétés 2.3) des patchworks dans des algorithmes plus efficaces. De telles améliorations peuvent être obtenues en adaptant les programmes actuels, toujours en Prolog.
- Tous les patchworks préservants considérés dans les exemples de cette section sont totaux et engendrent des super-groupes de taille 2 (révélés par $\overline{F}^2 = \overline{G}$). Nous avons bien sûr obtenu avec notre programme des exemples de super-groupes de tailles supérieures. Il reste à étudier les propriétés remarquables des super-groupes et leurs applications particulières éventuelles.
- Enfin, on a utilisé la notion de vue susceptible de révéler des groupes ou sous-groupes remarquables de collections d'objets. Nous en exhibons un exemple dans les chapitres suivants où les objets sont des solutions ou des "nogoods" de CSP. Cette idée gagnerait à être généralisée en exhibant bien sûr d'autres exemples. La généralité de cette approche devrait aider à y parvenir.

# 3 Les Symétries

On trouve dans la littérature un certain nombre de définitions sur les symétries. Elles dépendent souvent du contexte et entre autres du type d'élimination choisie. Tous s'accordent cependant pour dire qu'une symétrie possède forcément une propriété de bijection sur l'ensemble des solutions, mais dans la pratique cette seule définition englobe un bien trop grand nombre de fonction dont la plupart n'ont pas une réelle signification. Si on a $n$ solutions par exemple on aura $n!$ fonctions bijectives possibles, mais parmi elles un très grand nombre ne peut être caractérisé qu'en connaissant déjà l'ensemble des solutions. Donc dans la pratique on ne s'intéressera qu'à celles pouvant être définies *a priori*, par exemple une permutation de deux variables ou de deux valeurs. Nous donnons par cette section un certain nombre d'exemple, de définitions, un état de l'art sur l'élimination de symétries et enfin une implantation d'une d'entre elles, la SBDS.

## 3.1 Exemples de symétries dans des problèmes CSP

Nous présenterons dans cette partie quelques exemples de problème possédant des symétries classiques. Commençons par quelques exemples élémentaires :

**Example 3.1** [symétrie de variable] Soit un problème ayant pour unique contrainte : $X + Y + 2 * Z = 6$. Dans cette équation $X$ et $Y$ jouent le même *rôle*, si $(a, b, c)$ est une solution du triplet $(X, Y, Z)$ alors $(b, a, c)$ est aussi une solution. On parlera de symétrie de variable entre $X$ et $Y$ car la permutation de ces deux variables dans chaque triplet laisse invariant l'ensemble des triplets solutions.

**Example 3.2** [symétrie de valeur] Soit un problème ayant pour unique contrainte : $(X-1)*(X-2)*Y = 0$. Les solutions sont de la forme $(1, b)$, $(2, b)$ et $(a, 0)$, quelque soit $a$ et $b$. Si on permute pour chacune de ces solutions les valeurs 1 et 2 on obtient toujours une solution. Donc on considérera que l'on a une symétrie de valeurs entre 1 et 2.

Voici à présent quelques exemples classiques de problème avec symétries ([8]) :

**Example 3.3** [golfeur sociable] On désire organiser un tournoi de golf avec $n$ joueurs, durant $w$ semaines, chaque semaine ils sont répartis en $g$ groupes de $s$ joueurs $(n = g * s)$. Ce problème contient énormément de configurations équivalentes :
- en permutant les joueurs au sein d'un même groupe, pour une semaine donnée $\phi_s$ ;
- en permutant les groupes au sein d'une même semaine$\phi_g$ ;
- en permutant les semaines d'une solution$\phi_w$ ;
- en permutant les joueurs d'une solution$\phi_p$.

En conjuguant ces symétries on obtient un groupe de taille $(s!)^{gw}(g!)^w w!(gs)!$. Si on prend l'instance $g = 5$, $s = 3$ et $w = 7$, on aura environ $4 \cdot 10^{57}$ configurations symétriques à une même configuration.

**Example 3.4** [carré magique] Sur un échiquier de taille $n$ on place les nombres de 1 à $n^2$ de sorte que pour chaque colonnes, chaque lignes et pour les deux diagonales principales la somme des nombres correspondants soient la même. Il existe 7 symétries dites géométriques, qui sont dues à la structure du problème et qui correspondent aux transformations géométriques laissant invariant le carré (les symétries selon les axes horizontale, verticale et des deux diagonales, ainsi que les rotations à 90°, 180° et 270°). Il existe d'autres symétries liées aux équations arithmétiques, moins évidentes à trouver.

## 3.2 Définitions et liens avec les fonctions préservantes

Nous nommerons *symétrie* une fonction préservante conservant les solutions. Pour caractériser ces symétries on utilisera les fonctions de symétries qui peuvent avoir des définitions plus ou moins large et englobant plus ou moins de fonctions. Dans le cas le plus large une fonction de symétrie sera n'importe quelle bijection sur l'ensemble des solutions. Cependant cette définition est bien trop large et ne nous permet pas une étude systématique car elle s'appuie sur la connaissance de l'ensemble des solutions. On propose donc ici une définition se restreignant aux bijections sur les affectations.

**Definition 3.1** [restreinte] Soit un problème $\mathcal{P} = (V, C, D)$, de taille $n$. Soit $\sigma$ une bijection sur $Aff(V, D)$, avec $Aff(V, D)$ l'ensemble des affectations possibles ($(v \to d) \in Aff(V, D)$ signifie qu'on attribue la valeur $d$ à la variable $v$). Si pour toute solution $s$ de $\mathcal{P}$, telle que $s = \{(v_i \to d_i)\}_{i \in [1 \ldots n]}$, on a $s' = \{\sigma(v_i \to d_i)\}_{i \in [1 \ldots n]}$, avec $s'$ solution de $\mathcal{P}$, alors $\sigma$ est une fonction de symétrie du problème.

Cette définition nous permet de caractériser un très grand nombre de symétries. Donnons quelques sous-classes de cette ensemble de symétries parmi les plus souvent rencontrées.

**Definition 3.2** [symétrie de variable] Soient $\pi$ une permutation sur l'ensemble des variable $V$, et $\sigma$ une fonction de symétrie telle que $\forall (v \to d) \in Aff(V, D)$, $\sigma(v \to d) = (\pi(v) \to d)$. On dira que $\sigma$ est une fonction de symétrie de variable sur $V$.

On définit de même les symétries de valeurs :

**Definition 3.3** [symétrie de variable] Soient $\pi$ une permutation sur l'ensemble des variable $D$, et $\sigma$ une fonction de symétrie telle que $\forall (v \to d) \in Aff(V, D)$, $\sigma(v \to d) = (v \to \pi(d))$. On dira que $\sigma$ est une fonction de symétrie de valeur sur $D$.

## 3.3 Etat de l'art sur l'élimination de symétries

Comme on a pu le constater plus haut un problème donné peut avoir un très grand nombre de symétries. Concrètement cela signifie qu'à partir d'une solution on peut retrouver tout un ensemble de solutions à l'aide des fonctions de symétries. Dans ce cas on voudrait éviter la recherche de ces solutions par le solveur de contraintes.

De façon plus générale on aimerait que pour une affectation $v \to d$ on n'élimine dans la suite de la recherche toutes les affectations symétriques $\sigma(v, d)$. Ainsi on ne se limite pas uniquement aux solutions, on coupe dans l'arbre de recherche les branches inutiles et donc même pour un problème insatisfiable la recherche sera beaucoup plus directe.

### 3.3.1 Elimination statique

**Modélisation** Les symétries sont liées très fortement à la modélisation du problème, donc en la modifiant on peut parfois les éliminer directement . Dans le problème 3.3 on peut modéliser le problème sous la forme de liste de liste de liste :

$$\begin{aligned} Solution = &\ [Semaine1, Semaine2, \ldots] \\ = &\ [[Groupe1, Group2 \ldots], \ldots] \\ = &\ [[[p_1, p_2, \ldots], \ldots], \ldots] \end{aligned}$$

Dans cette modélisation toutes les symétries évoquées sont présentes, les variables représentent chacun des joueurs. On peut cependant choisir une autre représentation qui représente les groupes comme des ensembles. On attribue un nombre pour chacun des groupes d'une semaine, les variables représentent à présent l'appartenance d'un joueur à un des groupes :

$$Solution = [[Joueur1\_dans\_groupe\_G, \ldots], \ldots]$$

Dans ce cas la permutation à l'intérieur d'un groupe n'a plus de sens, on a éliminé la symétrie $\phi_s$. Cependant il n'est pas toujours aisé de modifier la représentation ou elle a un temps de résolution bien plus élevées que la modélisation initiale.

**Ajout de contraintes statiques** La méthode consiste à ajouter des contraintes au problème initial qui élimine d'emblée certaines symétries. Dans l'exemple 3.2 on peut ajouter la contrainte $X \neq 1$. Les solutions pour lesquelles $Y \neq 0$ n'auront plus de solutions symétriques. Entre les sous-ensembles de solutions de types $(1, y)$ et $(2, y)$ on a choisi le premier ensemble sachant que l'autre peut facilement être retrouver par la fonction de symétrie. Dans certains cas on peut imposer une relation d'ordre sur un ensemble de variables symétriques.

L'ajout de ces contraintes permet, sans modifier la modélisation, d'éliminer des symétries sans trop de coût. Cependant elles ne permettent pas en général d'éliminer toutes les symétries.

**Modelling Matrix** Un certain nombre de problème CSP peuvent être modéliser sous forme matricielle. Prenons pour exemple le problème du *round robin tournament* ([8]). On a $n$ équipes jouant pendant $n - 1$ semaines, chaque semaines étant divisée en $n/2$ périodes, on distingue les cas où une équipe joue à la maison ou à l'extérieur. Chaque équipe joue avec toutes les autres une fois, joue une fois par semaine et au plus deux fois dans la même période. On peut modéliser ce problème sous la forme d'une matrice de deux dimensions, avec les semaines pour les colonnes et les périodes pour les lignes, chaque variable correspondant à un match (par exemple si l'équipe $i$ rencontre chez elle l'équipe $j$ la variable vaudra $ij$). On a de toute évidence une symétrie sur les semaines et sur les périodes, donc les colonnes ou les lignes peuvent permuter.

Pour casser ces symétries un ordre lexicographique peut donc être imposé, soit sur les colonnes soit sur les lignes, voire sur les deux. En effet, comme il est démontré dans [5], pour une modèle matriciel de dimension deux, ayant des symétries totales sur les colonnes et sur les lignes, chaque classe d'équivalence (regroupant les solutions symétriques) contient au moins un élément où les lignes et les colonnes sont lexicographiquement ordonnées. On montre également l'existence d'une solution anti-lexicographiquement ordonnée. Par contre il n'y a pas forcément de solution où une des dimensions est lexicographiquement ordonnée et l'autre anti-lexicographiquement ordonnée. On peut tout aussi bien gérer les symétries non totales en regroupant certaines colonnes ou lignes, et donner un ordre lexicographique à l'intérieur de ces ensembles.

Cette méthode se généralise à des dimensions supérieures à deux et il est montré que pour un modèle matriciel, dont chaque dimension possède des symétries dans une matrice de dimension $n$, où $n \geq 1$, chaque classe d'équivalence possède un élément où cette matrice est lexicographiquement ordonnée.

De même on peut gérer les symétries de valeurs en posant le problème dual, où chaque valeur (devenue variable) devient un vecteur de 'variables' (devenues valeurs). De plus il est possible, théoriquement, de casser toutes les symétries en imposant un ordre lexicographique spécifique qui tient compte de toutes les symétries. Mais c'est souvent fastidieux, et c'est applicable que dans des cas particuliers.

### 3.3.2 Elimination dynamique

Comme nous avons vu plus haut il n'existe pas de méthode d'élimination statique des symétries qui soit générique pour tous les problèmes. Pour éliminer correctement une symétrie il faut être sûr lorsqu'on développe l'arbre de recherche que les affectations symétriques des affectations déjà essayées ne seront jamais faites. Les éliminations doivent donc suivre au plus près de l'arbre de recherche donc dynamiquement. Les deux méthodes décrites dans ce chapitre répondent à ces critères et ont obtenu des résultats probant sur des problèmes difficiles.

**Symmetry Breaking During Search** Cette méthode a été proposée par R. Backofen et S. Will [1], puis développée par Ian Gent et Barbara Smith [7]. L'idée directrice est de rajouter des contraintes dynamiquement, pendant la résolution, afin que les symétries des solutions issues du point de choix courant soient supprimées. Elle vérifie les propriétés suivantes :

1. toutes les symétries données par l'utilisateur sont cassées ;
2. l'heuristique du parcours de l'arbre de choix est préservée.

Pour la suite nous avons besoin d'introduire des nouvelles notations afin de formaliser l'arbre de recherche. Nous noterons $\tilde{\sigma}$ la fonction associée à $\sigma$ mais définie sur les contraintes d'affectations (si $\sigma(v \to d) = (v' \to d')$ alors $\tilde{\sigma}(v = d) = (v' = d')$ et $\tilde{\sigma}(v \neq d) = (v' \neq d')$). Soit $\mathcal{P}$ un problème CSP, $C$ un ensemble de contraintes, on notera $\mathcal{P}' = \mathcal{P} \cup C$ un problème CSP, équivalent au problème $\mathcal{P}$ auquel on a ajouté les contraintes de $C$. On modélisera chaque noeud de l'arbre de recherche par le triplet $N = (\mathcal{P}, \Sigma, C_{aff})$, où $\Sigma$ est l'ensemble des symétries de $\mathcal{P}$ données dans le problème, et $C_{aff}$ l'ensemble des contraintes d'affectation posées depuis la racine. Ainsi le premier noeud de notre arbre de recherche sera de la forme $N_0 = (\mathcal{P}, \Sigma, \{\})$, $\Sigma$ contenant l'ensemble des symétries données par l'utilisateur. L'algorithme pour développer l'arbre de recherche est donné figure 1.

1. *Initialisation :* $Noeud\_courant = N_0$ *;*
2. $Noeud\_courant = (\mathcal{P}, \Sigma, C_{aff})$ *;*
3. *on choisit une affectation possible dans* $\mathcal{P}$ *:* $v \to d$ *;*
4. $\mathcal{P}^a = \mathcal{P} \cup \{v = d\}$ *;* $\Sigma^a \subset \Sigma$*, tel que* $\forall \sigma \in \Sigma^a$*,* $\sigma$ *symétrie de* $\mathcal{P}^a$ *;*
   $C^a_{aff} = C_{aff} \cup \{v = d\}$ *;* $N^a = (\mathcal{P}^a, \Sigma_a, C^a_{aff})$ *;*
5. $Noeud\_courant = N^a$*, on continue en* 2) *avec ce nouveau noeud;*
6. *une fois ce sous-arbre complètement développé on revient en arrière :* $Noeud\_courant = (\mathcal{P}, \Sigma, C_{aff})$ *;*
7. $C^b = \{\bigwedge_{aff \in C_{aff}} \tilde{\sigma}(aff) \Rightarrow \tilde{\sigma}(v \neq d)\}_{\sigma \in \Sigma}$ *;* $\mathcal{P}^b = \mathcal{P} \cup \{v \neq d\} \cup C^b$*;* $N^b = (\mathcal{P}^b, \Sigma, C_{aff})$ *;*
8. $Noeud\_courant = N^b$*, on continue en* 2) *avec ce nouveau noeud.*

FIG. *10* – **Algorithme SBDS.**

L'arbre de recherche est donc un arbre binaire (voir figure 2), la première branche pose $v = d$ et cherche les symétries restantes parmi les symétries initiales et la deuxième branche pose $v \neq d$ et un ensemble de contraintes conditionnelles $C_b$. Ces dernières contraintes permettent de ne pas retomber sur un ensemble d'affectations symétriques de la première branche.

Dans l'étape 4 on doit tester toutes les symétries de $\Sigma$ afin de savoir si elles restent des symétries pour $\mathcal{P}^a$. Donc pour chaque symétrie $\sigma \in \Sigma$ on calcule l'ensemble $C^a$ des affectations symétriques de $C^a_{aff}$, $C^a = \{\tilde{\sigma}(aff)\}_{aff \in C^a_{aff}}$, si $\mathcal{P}^a \cup C^a$ conduit par propagation à une insatisfaisabilité, on retire $\sigma$, sinon on la garde.

$$\begin{tikzcd}[column sep=small]
 & N = (\mathcal{P}, \Sigma, C_{aff}) \arrow[dl, "(v = d)"'] \arrow[dr, "(v \neq d) \cup C^b"] & \\
N^a = (\mathcal{P} \cup \{v = d\}, \Sigma^a, C_{aff} \cup \{v = d\}) & & N^b = (\mathcal{P} \cup \{v \neq d\} \cup C^b, \Sigma, C_{aff} \cup \{v \neq d\})
\end{tikzcd}$$

FIG. 11 – Un noeud de l'arbre de recherche lorsqu'on applique la SBDS.

Cette technique permet de casser à coup sûr les types de symétries telles qu'on les a définie, cependant, dans un problème avec de nombreuses symétries on atteint très vite une saturation sur le nombre de contraintes à gérer. Une solution consiste à prendre des sous-ensembles de symétries [10] au risque de ne pas casser toutes les symétries. Une autre technique s'appuie sur les propriétés de groupe de ces symétries [6], ainsi on n'est pas obligé de traiter chaque symétrie une à une car on est capable de calculer uniquement les informations utiles à l'élimination.

**Symmetry Breaking via Dominance Detection (SBDD)** Au lieu de rajouter des contraintes une des méthodes peut consister à vérifier à chaque point de choix si l'on n'est pas en présence d'un cas symétrique déjà exécuté. La méthode SBDD, proposée par *Fahle et al.* dans [4], s'appuie sur des relations de domination entre ces différents points de choix pour éliminer les symétries.

Soit $X = \{x_1 \ldots x_n\}$, l'ensemble des variables d'un problème. $D(x)$ représentera le domaine de la variable $x$. On notera alors $P^c = (D^c(x_1), \ldots, D^c(x_n))$ le tuple courant au point de choix $c$, appelé aussi le *pattern*.

Soit $c$ et $c'$ deux points de choix, alors :

$$P^{c'} \; inclut \; P^c \; si, \; et \; seulement \; si, \; \forall x \in X : \; D^c(x) \subseteq D^{c'}(x)$$

De plus si $MD^c = D^c(x_1) \times \ldots \times D^c(x_n)$, étant donné une fonction $\phi$ de symétrie, $\phi : \; MD^c \to MD^c$, on dit alors que :

$$P^{c'} \; domine \; P^c \; par \; rapport \; à \; \phi \; ssi \; \phi(P^c) \subseteq P^{c'}.$$

On notera cela $P^c \sqsubseteq P^{c'}$.

Soit un successeur $c'$ de $c$ dans l'arbre de recherche, il est clair que $P^{c'} \sqsubseteq P^c$.

Cette méthode exige de stocker les données sur les différents points de choix dans une base, et d'établir une fonction $\Phi : (P^{\Delta}, P^{\Box}) \to \{faux, vrai\}$, qui prend la valeur vraie ssi $P^{\Delta}$ est dominé par $P^{\Box}$ par rapport à *une* symétrie $\phi$. La base de donnée $T$ de tous les parterns risque d'être très grande si on stocke tous les points de choix. Les auteurs proposent pour limiter ce phénomène d'appliquer la recherche en profondeur d'abord. En effet on remarque que si un point de choix $c'$ est dominé par le point de choix $c$ par rapport à une symétrie, alors $c'$ est aussi dominé par le père de $c$. Il n'est donc pas nécessaire d'enregistrer le pattern associé à $c$ si on sait que tous les points de choix développés à partir de son père ont été complètement parcourus (voire figure 12).

SBDD a l'avantage de ne pas toucher aux contraintes, l'overhead ne vient donc que de la gestion du store des patterns. Lorsqu'on a un problème avec beaucoup de symétries SBDD est alors bien plus avantageux que SBDS, car dans le premier cas on va trouver les premiers patterns relativement rapidement et ensuite

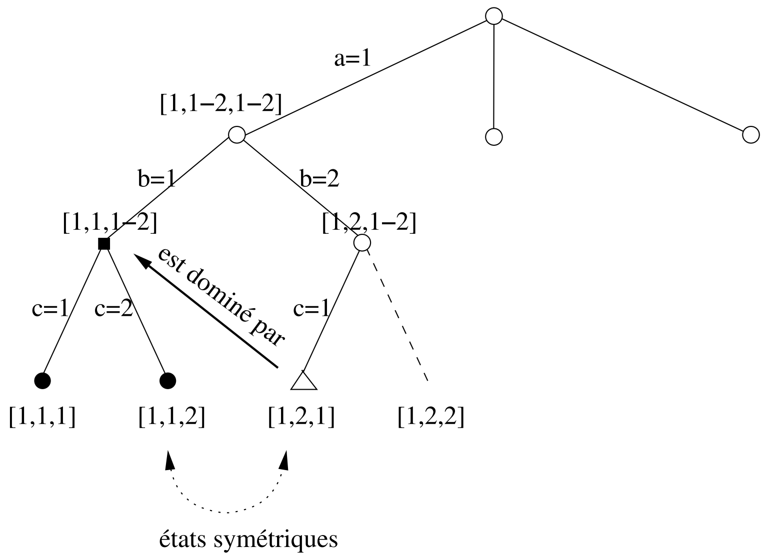


FIG. 12 – Exemple d'un élaguage des symétries par SBDD. On suppose que l'on a 3 variables [a,b,c], de domaines [[1..3] ;[1..2] ;[1..2]]. De plus on suppose que b et c sont symétriques. Les ronds blancs représentent les points de choix non complétement développés, les ronds noirs les états finaux déjà trouvés, les carrés noirs les points de choix qui figurent dans le store des patterns, et le triangle blanc le point de choix courant. La stratégie de parcours est ici la profondeur d'abord.

on pourra éliminer les symétries, dans le deuxième cas beaucoup de contraintes vont être rajoutées surtout vers la fin de la recherche, et risque de ralentir énormément la résolution. C'est le cas entre autre du problème du golfeur sociable (cf. exemple 3.3).

A contrario SBDD est moins performant pour les problèmes avec peu de symétries, comme le problème des reines, la gestion du store devient trop lourde par rapport au bénéfice qu'il apporte. De plus la pertinence de cette technique dépend de la méthode de recherche, la recherche en profondeur d'abord étant la plus efficace pour éliminer les symétries au plus tôt.

## 3.4 Implantation de la SBDS en GNU Prolog

Comme nous l'avons vu la SBDS a l'avantage de ne pas être dépendant de la méthode de parcours de l'arbre de recherche (first_fail, most_constrained, random, ...), elle n'intervient qu'à chaque point de choix. Il nous a donc semblé intéressant de construire une fonction d'élimination de symétrie, dans le solveur de contraintes GNU Prolog [3], la plus simple et la plus transparente possible pour l'utilisateur. Nous aborderons dans un premier temps le problème de la spécification des symétries que doivent fournir l'utilisateur et ensuite l'implantation de l'algorithme décrite à la section 3.3.2.

### 3.4.1 La spécification des symétries

Pour spécifier une symétrie on a besoin d'associer à chaque affectation possible (un couple $(Var, Val)$) une affectation symétrique. Le prédicat *calcule_sym/3* pre-

nant en entrée le nom du prédicat de la symétrie, défini par l'utilisateur, et le couple $(Var, Val)$ fournira en sortie le couple $(VarSym, ValSym)$ symétrique.
**Exemple :** Prenons l'exemple des 8 reines, $V = \{v_1 \dots v_8\}$ représente les colonnes du damier, $D = [1 \dots 8]$. Supposons que l'utilisateur a spécifié la symétrie verticale *vert*, échangeant la colonne 1 avec la 8, la 2 avec la 7, etc..., et on veut connaître l'affectation symétrique de $v_1 = 1$, alors on fait l'appel :

```
calcule_sym(vert,(1,1),(VarSym,ValSym)).
```

qui donnera le résultat $(VarSym, ValSym) = (8, 1)$.

L'écriture du prédicat de symétrie peut soit être faite directement par l'utilisateur soit, dans certains cas, peut être générée par un prédicat *generateur_sym/2*.
**Exemple :** Reprenons le cas des 8 reines, le prédicat de la symétrie verticale s'écrira :

```
vert((Var,Val),(VarSym,Val)):-
       VarSym is 9-Var.
```

On a également une symétrie de rotation de $+90°$, *rot90* ($\forall i,\ v_i = j$ a pour image, par $rot90$, $v_j = 9 - i$) qui s'écrira :

```
rot90((Var,Val),(Val,ValSym)):-
       ValSym is 9-Var.
```

On notera que $Var$ représente l'indice de la variable, i.e sa position dans la liste donnée par l'utilisateur, il est donc crucial que cette liste corresponde bien au modèle. De plus toutes les symétries doivent être de la forme *nom_sym((Var, Val),(VarSym, ValSym)*. Par contre il n'est pas nécessaire de traîter le cas où un couple reste invariant.
**Exemple :** Soit $V = \{v_1 \dots v_n\}$ un ensemble de variables, supposons qu'il existe une symétrie de variable entre $v_1$ et $v_2$, on posera alors seulement ces deux prédicats :

```
sym((1,Val),(2,Val)).
sym((2,Val),(1,Val)).
```

Certaines symétries peuvent être laborieuses à écrire ou dans certains cas on peut éprouver le besoin de les paramétrer (par exemple en fonction du nombre de variables). C'est dans ce but qu'une fonction génératrice de symétries a été implantée.
**Exemple :** Soit $V = \{v_1 \dots v_n\}$, on veut construire une symétrie totale de variable sur l'ensemble $\{v_1, v_2, v_3\}$ on écrira :

```
generateur_sym([sym_var([1,2,3])],Liste_Sym)
```

qui générera tous les prédicats spécifiant cette symétrie, par exemple la symétrie *sym_var_1_2/2* entre la variable 1 et 2.

Le prédicat *sym_var(L)* indique au générateur de symétries que l'on a une symétrie de variable sur $L$, *Liste_Sym* contient l'ensemble des noms des prédicats des symétries générées (plusieurs symétries sont générées car on a une symétrie de variable entre $v_1$ et $v_2$, $v_1$ et $v_3$, etc...). L'utilisateur peut créer lui-même un tel prédicat, qui générerait par exemple toutes les symétries pour le problème des *n-reines*. A terme il serait également utile de créer un prédicat générant le groupe entier des symétries à partir de celles fournies par l'utilisateur.

### 3.4.2 La nouvelle fonction de labeling

Une fois que l'utilisateur a spécifié tous ces prédicats de symétrie il n'a plus qu'à rassembler tous leurs noms dans une liste utilisée comme argument de la nouvelle fonction de labeling :

```
fd_labeling_sbds(ListeVariables,ListeSymetries,ListeOptions).
```

Dans la liste des options on peut spécifier, comme dans un labeling normal de GNU Prolog, l'ordre d'instanciation des variables et des valeurs. Cette fonction va ensuite appeler la fonction principale d'énumération dont on donne une version très simplifiée (sans certaines optimisations entre autres) :

```
sbds([],_,_,_):-!.
sbds(Lvar,Lsym,Laff,LOptions):-
    % On choisit une affectation selon les méthodes
    % passées dans LOptions
      choix_affectation(Lvar,Var,Val,LOptions),
      !, % On veut contrôler nous même le retour arrière

      (
      % Affectation
        Var#=Val,
      % On retire Var de la liste des variables
        retirer_liste(Lvar,Var,Lvar_nv),
      % On ajoute la nouvelle affectation dans la liste
        ajouter_liste(Laff,(Var,Val),Laff_nv),
      % Réduction des symétries superflues après affectation
        reduction_ensemble_symetries(Laff_nv,Lsym,Lsym_nv),
      % On descend dans l'arbre de recherche
        sbds(Lvar_nv,Lsym_nv,Laff_nv,LOptions)

      ; % Backtrack après l'échec d'une affectation

        Var##Val,
      % On pose les contraintes pour casser les symétries
        casser_symetries((Var,Val),Lsym,Laff),
      % On essaye une nouvelle affectation
        sbds(Lvar,Lsym,Laff,LOptions)
      ).
```

La disjonction ';' permet de créer un point de choix dans l'arbre de recherche afin de rajouter dans la deuxième branche les contraintes d'éliminations. La coupure après *choix_affectation* interdit tout retour arrière sur ce prédicat, dans le cas contraire on perdrait les contraintes rajoutées dynamiquement.

Le prédicat *reduction_ensemble_symetries* calcule pour chaque symétrie l'affectation symétrique de l'affectation courante et teste si elle est consistante avec l'affectation courante.

Dans la deuxième partie de la disjonction on pose les contraintes nécessaires avant d'essayer une nouvelle affectation de la variable courante. Comme nous l'avons vu dans le section 3.3.2, on a besoin de poser les contraintes conditionnelles de type $sym(Laff) \Rightarrow (VarSym\#\#ValSym)$.

```
casser_symetries(_,[],_).
casser_symetries((Var,Val),[Sym|LSym],Laff):-
    % calcule l'affectation symétrique de (Var,Val) par Sym
      calcule_sym(Sym,(Var,Val),(VarSym,ValSym)),
    % calcule l'affectation symétrique de Laff
    % LCtes est l'équivalent de LaffSym sous forme de contraintes
      affectation_sym(Sym,Laff,LaffSym,LCtes),
    % Nombre d'affectation dans l'état courant
      length(Laff,Nb_aff),
```

```
% La nouvelle variable contraintes Nb_Ctes_true aura pour valeur
% le nombre de contraintes de LCtes vraies
  fd_cardinality(LCtes,Nb_Ctes_true),
% Si toutes les contraintes symétriques de Laff sont vraies
% alors on peut retirer la valeur ValSym à VarSym
  (Nb_Ctes_true#=Nb_aff)#==>(VarSym#\=ValSym),
  casser_symetries((Var,Val),LSym,Laff).
```

*LCtes* est de la forme *[VarSym1#=ValSym1,...]*, on utilise ensuite une variable intermédiaire *Nb_Ctes_true* car la contrainte d'implication # ==> ne travaille que sur des booléens. La contrainte *fd_cardinality* est une contrainte globale GNU Prolog, c'est une des rares contraintes non ISO et la conversion pour un autre solveur de contraintes reste assez simple.

# 4 Détection de symétries à l'aide d'outils visuels

## 4.1 La détection de symétrie

La détection de symétries est un problème important et pas toujours évident dans les CSP. De plus elle dépend du type de symétries que l'on recherche. Il existe au moins trois façon de les détecter :

1. *Structurellement* : On analyse la structure du problème, en particulier la structure de donnée. Dans le problème du carré magique (exemple 3.4), si on choisit de représenter une case par une variable les symétries de type géométrique sont assez évidentes.
2. *Analytiquement* : L'étude systématique des contraintes et des variables et leur domaine permettent, dans certains cas, de trouver assez rapidement des symétries du problème. Pour une symétrie de variable, par exemple, on peut calculer tous les ensembles de contraintes correspondant aux permutations sur les variables. Si un de ces ensembles est équivalent, à un ordre près sur les contraintes, à l'ensemble initial du problème alors il existe une symétrie de variable. Cependant cette méthode est laborieuse face à de gros problème avec beaucoup de variable et de contraintes. Il existe aussi des méthodes constructives où à chaque contrainte du problème rajoutée on réduit les classes d'équivalence (voir [2]) mais les règles d'inférence ne sont pas toujours évidentes.
3. *Visuellement* : L'ensemble des symétries d'un problème peut très vite atteindre des tailles inexploitables pour un système analytique. Les outils visuels peuvent alors guider l'utilisateur afin de détecter un certain nombre de ces symétries, soit *a priori*, par des graphes de contraintes par exemple, soit *a posteriori*, ce qui, dans certains cas, exige d'avoir un bon traceur pour les solveurs.

Dans la section suivante on étudiera le cas d'une visualisation a posteriori s'appuyant sur le nombre de solution pour chaque occurrence $(Variable, Valeur)$.

## 4.2 Vue Variables/Valeurs/Nombres de solutions

### 4.2.1 Description de l'heuristique

On part de l'idée que si deux couples $(Variable, Valeur)$ sont symétriques alors chacun des deux éléments apparaît le même nombre de fois dans l'ensemble des solutions (si aucune de ces symétries n'a été cassée).

Proof. Soit le problème CSP $\mathcal{P} = (C, V, D)$. Soient $\sigma$ une symétrie de $\mathcal{P}$ sur l'ensemble des solutions $S$. Supposons que $v_i \to d_j$ et $v_m \to d_n$ ($\{v_i, v_m\} \in V$ et $\{d_j, d_n\} \in D$) soient symétriques par rapport à $\sigma$ ( $\sigma(v_i \to d_j) = v_m \to d_n$) et

$$S_1 = \{s | s \in S, v_i \to d_j \in s\}$$

$$S_2 = \{s | s \in S, v_m \to d_n \in s\}$$

or $\sigma(S_1) = S_2$ et $\sigma$ est bijective. On a bien :

$$|S_1| = |S_2|$$

□

On se propose donc dans un tableau, où chaque ligne représente une valeur et chaque colonne une variable, de calculer le nombre de fois qu'une variable $Var$ prend la valeur $Val$.

**Proposition 4.1** [Quelques propriétés liées à la visualisation (Variable,Valeur)]

1. Les colonnes représentent la distribution des solutions par rapport à chaque variable,
2. Les lignes représentent la distribution des solutions par rapport à chaque valeur,
3. Pour deux distributions identiques on peut faire l'hypothèse d'une symétrie de valeur ou de variables.
4. S'il y a une symétrie, telle définie dans la section 3.2, dans le problème elle est observable sur notre représentation variable/valeur (plus ou moins facilement selon l'ordre des données) ;
5. Si on a deux distributions différentes sur notre figure on est sûr que ces deux éléments ne sont pas symétriques.

### 4.2.2 Les avantages et les limites de ce type d'observation

Une des première remarque que l'on peut donner c'est que l'heuristique ne donne pas précisément la forme de la symétrie, si on a par exemple deux couples de deux variables $(x_1, x_2)$ et $(y_1, y_2)$ ayant chacune la même distribution on ne peut pas dire si on a la symétrie :

$$\sigma : (x_1, y_1) \to (x_2, y_2)$$

ou si on a deux symétries :

$$\sigma_x : x_1 \to x_2 \; et \; \sigma_y : y_1 \to y_2.$$

On a donc besoin d'une analyse supplémentaire pour déterminer de quelles symétries il s'agit. Nous cherchons actuellement à caractériser des symétries particulières dans ce type d'observation. Cette difficulté vient, entre autre, que l'observation des symétries variables/valeurs est très dépendante de l'ordre des données et, en générale, assez difficile.

Dans certains cas des distributions identiques ne correspondent pas à des symétries et demandent à être vérifiées (par exemple une symétrie totale sur des variables donnera une distribution identique des valeurs pour ces variables, voire exemple de

Schur). Cependant l'expérience montre que si on a un nombre suffisant de solutions il est très fréquent que l'heuristique ait un sens.

Le coût supplémentaire lors du traçage est plutôt faible, il suffit d'incrémenter la matrice la matrice variable/valeur chaque fois qu'on obtient une solution. Par contre dans le cas où on a peu de solutions l'heuristique est mise à défaut. L'adaptation de cette méthode aux échecs est donc nécessaire. Comme nous l'avons vu dans la section 2

L'étude ne se fait qu'a posteriori, donc si la résolution est trop longue ça limite son intérêt.

## 4.3 Différents exemples de vue

Les différentes vues présentées ici on était faites gâce à Discovery, développé par Ilog.

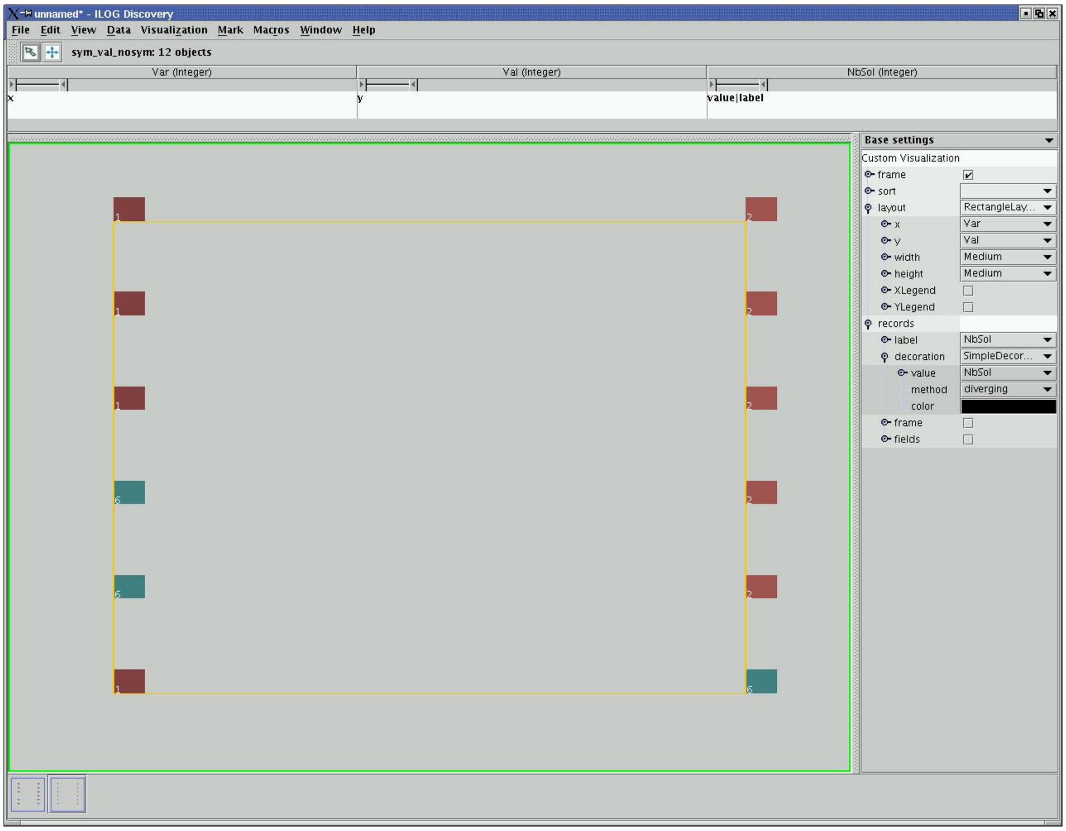


FIG. 13 – **Symétrie de valeur :** Exemple 3.2 $((X-1)*(X-2)*Y=0)$, . chaque variable a pour domaine $[0,5]$. Les lignes 2 et 3 sont identiques, elles représentent les valeurs 1 et 2, les deux colonnes représentent $X$ et $Y$.

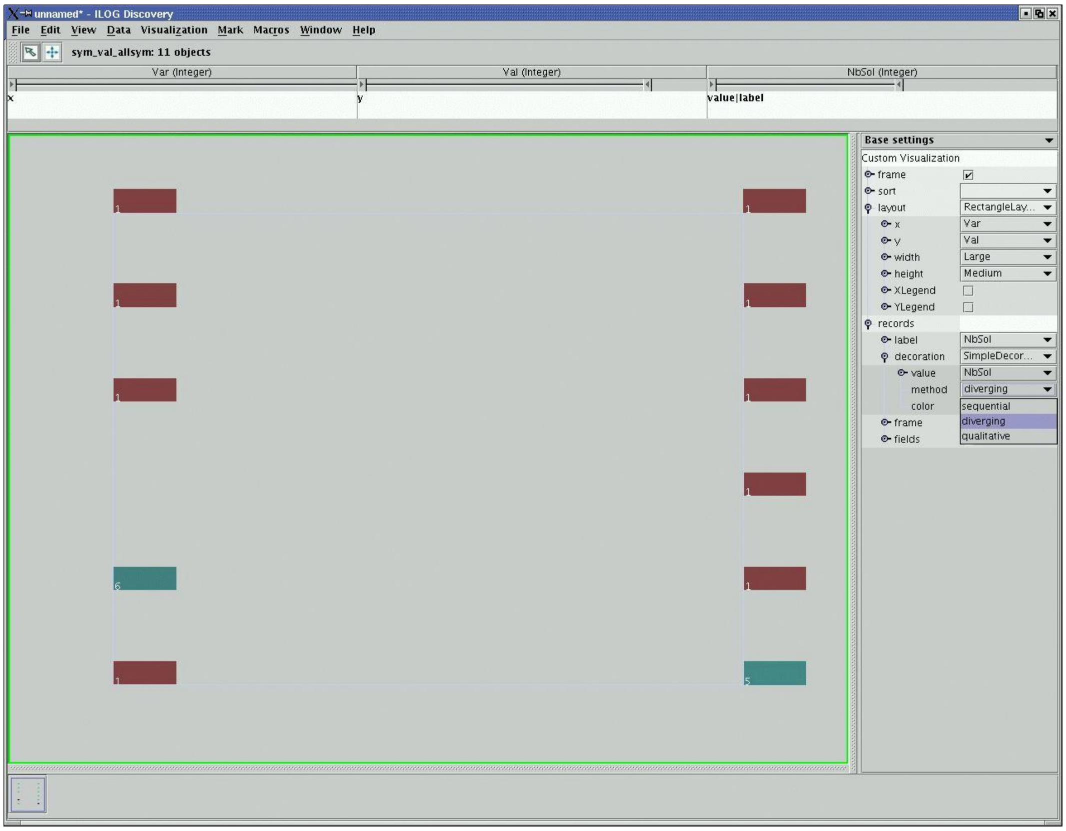


FIG. 14 – **Symétrie de valeur :** La symétrie de valeur a été éliminée, on ne remarque plus de distribution identique pour les valeurs 1 et 2.

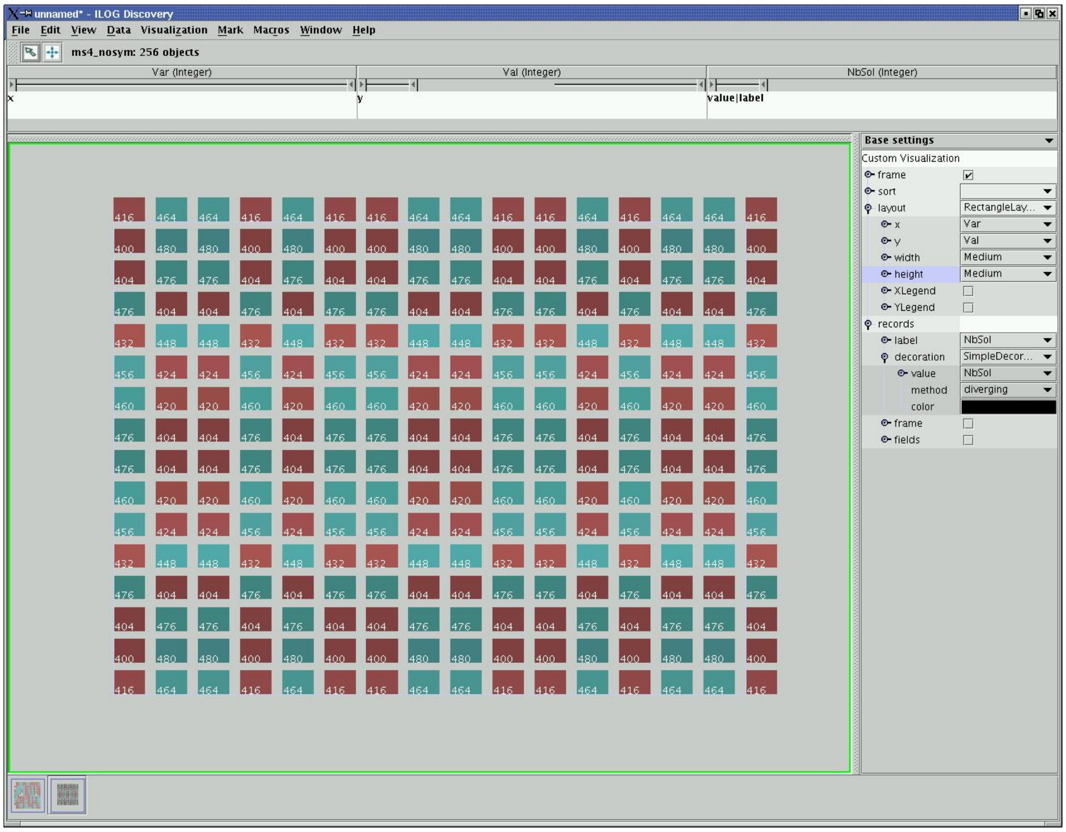


Fig. 15 – **Carré magique :** Exemple 3.4 de dimension $4x4$, les 16 variables sont représentées par les colonnes et les valeurs par les lignes. Ici aucune symétrie n'a été éliminée. On voit nettement un certain nombre de symétries entre les variables (colonnes identiques) et une symétrie de valeurs (la figure a une symétrie selon l'axe horizontale qui donne un idée de la symétrie du problème : si on échange la valeur 1 avec la valeur 16, la 2 avec la 15 …, le problème reste inchangé, on a bien une symétrie de valeur).

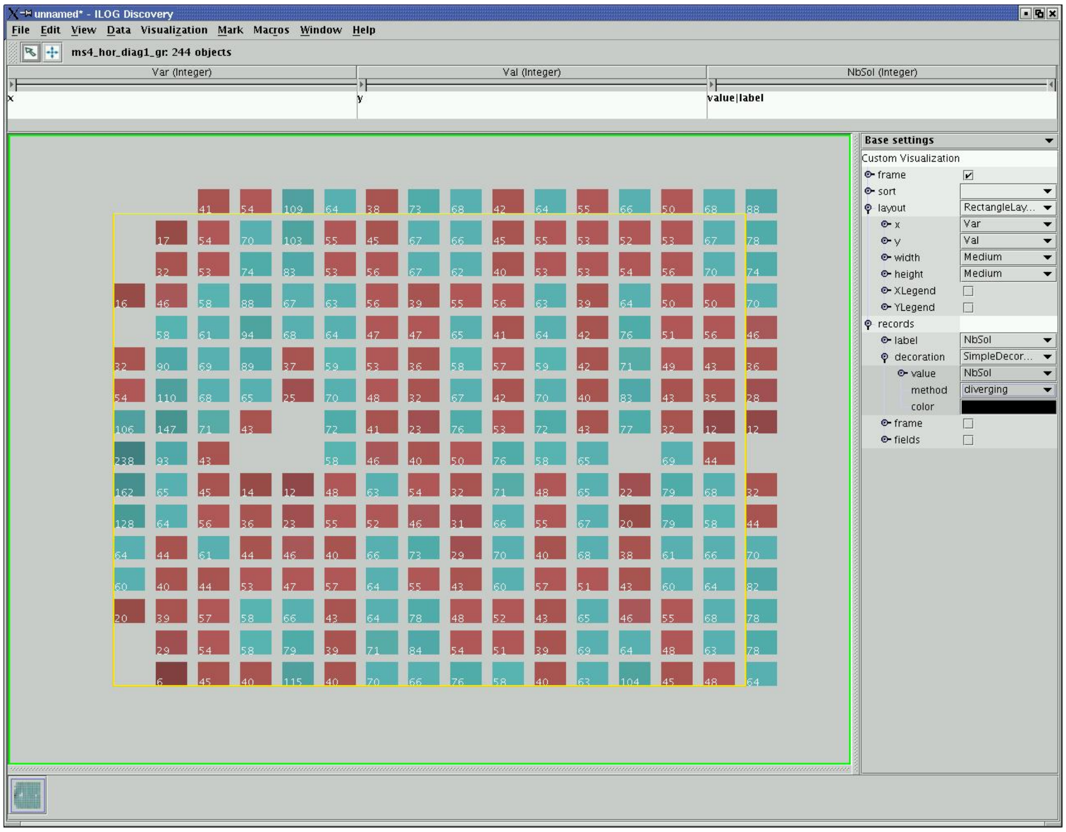


FIG. 16 – **Carré magique :** On a éliminé le groupe de permutation engendré par $(m1, s1)$. Bien que cela soit plus difficilement remarquable certaines colonnes sont toujours identiques, après étude on s'aperçoit qu'il s'agit de symétries arithmétiques.

## 5 Conclusion

Dans le soucis constant d'une optimisation des algorithmes de recherche des CSP, le problème des symétries occupe une place primordiale. Un certain nombre de méthodes d'élimination sont à présent au point et fournissent des améliorations indiscutables pour les résolutions. Cependant ces méthodes sont très dépendantes au type du problème, à sa taille ou encore au solveur de contraintes choisi. De plus les algorithmes d'élimination, dans le cas de problèmes contenant beaucoup de symétries, demandent encore à être améliorés et ce essentiellement pour les stratégies à adopter sur le choix des symétries à considérer, par exemple dans le cas d'une élimination partielle, sur l'ordre dans lequel se fait les valuations lors du parcours de l'arbre de recherche ou sur toute heuristique qui permet de réduire l'overhead dû à la prise en compte des symétries lors de la résolution. Notre étude s'attache donc à formaliser des propriétés sur les symétries qui permettraient par la suite de développer des outils visuels d'aide à la décision sur ces choix stratégiques. Dans l'état actuel de notre recherche il manque encore à faire une connexion entre les différentes étapes ( détection → élimination → visualisation → analyse). Le développement d'outil d'étude de groupe, d'élimination de symétries et de génération de trace de solveurs de contraintes (cf le traceur `Codeine`,`http ://contraintes.inria.fr/~langevin/codeine`) va nous permettre de préciser les besoins et d'affiner nos outils de visualisation.

## Références


[1] R. Backofen and S. Will. Excluding symmetries in concurrent constraint programming. *Workshop on Modeling and Computing with Concurrent Constraint Programming*, 1998. http ://citeseer.nj.nec.com/backofen98excluding.html.

[2] Jacqueline Chabrier and Eric Nespoulous. Détection constructive de symètries pour les csp entiers. *Journées Nationales de la résolution Pratique des problèmes NP-Complets (JNPC) - Lyon*, pages 47–54, June 1999.

[3] Daniel Diaz and Philippe Codognet. Design and implementation of the gnu prolog system. *Journal of Functional and Logic Programming*, 6-10, 2001.

[4] Torsten Fahle, Stefan Schamberger, and Meinolf Sellmann. Symmetry breaking. In T. Walsh, editor, *Contraint Programming, CP'01*, pages 93–107. Springer-Verlag, 2001. LNCS 2239.

[5] Pierre Flener, Alan M. Frisch, Brahim Hnich, Zeynep Kizitan, Ian Miguel, Justin Pearson, and Toby Walsh. Breaking row and column symmetries in matrix models. In T. Vidal and P. Liberatore, editors, *Proceedings of STAIRS 2002*. IOS Press, 2002.

[6] Ian P. Gent, Warwick Harvey, and Tom Kelsey. Groups and constraints : Symmetry breaking during search. In Hentenryck [9], pages 415–431. LNCS 2470.

[7] Ian P. Gent and Barbara M. Smith. Symmetry breaking in constraint programming. In Werner Horn, editor, *14th European Conference on Artificial Intelligence*, pages 599–603, Berlin, Allemagne, 24 et 25 Juillet 2000. ECCAI and German infortics society, IOS press.

[8] Ian P. Gent and Toby Walsh. Problem library for constraints. *site internet*. http ://4c.ucc.ie/ tw/csplib/.

[9] Pascal Van Hentenryck, editor. *Proceeding of the 8th Internationnal Conference of Constraint Programming (CP 2002)*, Ithaca, NY, USA, 2002. Springer. LNCS 2470.

[10] Iain McDonnald and Barbara Smith. Partial symmetry breaking. In *Constraint Programming, CP'02* [9].