UNIVERSITÉ LIBRE DE BRUXELLES
FACULTÉ DES SCIENCES
DÉPARTEMENT D'INFORMATIQUE

INFO-F-302 - Logique Informatique
Projet : Le jeu Pattern
et
Utilisation de MiniSAT

OOMS Aurélien, SONNET Jean-Baptiste

Année académique 2012 - 2013

Table des matières

Chapitre 1

Question 1 : Énumération 3,2

1.1 Problème et notation

1.1.1 Grille

Le problème est présenté sous forme d'une grille 3×3 contenant au max 2 contraintes par ligne ou colonne.

Soit une matrice 3×3,

$$\begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} \\ x_{1,0} & x_{1,1} & x_{1,2} \\ x_{2,0} & x_{2,1} & x_{2,2} \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

où chacune des cases $x_{i,j}$ prendra potentiellement une des 2 valeurs : $\{-1, 1\}$, respectivement le blanc, le noir. Les cases sont initialisées avec la valeur 0.

On aura par exemple comme problème à résoudre :

$$\begin{array}{cc} & \begin{matrix} 2 & 1 & 2 \end{matrix} \\ \begin{matrix} 1 \\ 1\,1 \\ 2 \end{matrix} & \begin{bmatrix} 1 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{array} \quad \begin{array}{|c|c|c|} \hline \blacksquare & \square & ? \\ \hline ? & ? & ? \\ \hline ? & ? & ? \\ \hline \end{array}$$

Selon les contraintes précisées, la solution devra donner pour toutes les cases inconnues une valeur de 1 ou -1 :

$$\begin{array}{cc} & \begin{matrix} 2 & 1 & 2 \end{matrix} \\ \begin{matrix} 1 \\ 1\,1 \\ 2 \end{matrix} & \begin{bmatrix} 1 & -1 & -1 \\ 1 & -1 & 1 \\ -1 & 1 & 1 \end{bmatrix} \end{array} \quad \begin{array}{|c|c|c|} \hline \blacksquare & \square & \square \\ \hline \blacksquare & \square & \blacksquare \\ \hline \square & \blacksquare & \blacksquare \\ \hline \end{array}$$

1.1.2 Variables

Chaque case est représentée par une variable x, de sorte que la case à la i^{eme} ligne et à la j^{eme} colonne se note $x_{i,j}$.

On a que $(i,j) \in \{1,2,3\}^2$ et que chaque variable peut prendre comme valeur $v \in \{-1,0,1\}$ soit :

$$X = \{x_{i,j} | (i,j) \in \{0,1,2\}^2, v \in \{-1,0,1\}\}$$

1.1.3 Sémantique

On a pour sémantique :
- $x_{i,j}$ est **vrai** *si et seulement si* la case (i,j) a pour valeur $v = 1$
- $x_{i,j}$ est **faux** *si et seulement si* la case (i,j) a pour valeur $v = -1$

1.1.4 Codage

Pour miniSAT, il faut coder les propositions par des entiers.
La taille d'une grille est de 3×3, on codera en base 3.
Pour $x_{i,j}$ on aurait par exemple le codage : $(i \times 3) + j$

1.2 Contraintes

1. (trivial) **Pour chaque** ligne, s'il existe une bande, **toutes** les cases ne sont pas fausses.

$$\bigwedge_{i \in \{0,1,2\}} \neg \left(\bigwedge_{j \in \{0,1,2\}} \neg x_{i,j} \right)$$

soit,

$$\bigwedge_{i \in \{0,1,2\}} \left(\bigvee_{j \in \{0,1,2\}} x_{i,j} \right)$$

idem pour chaque colonne.

$$\bigwedge_{j \in \{0,1,2\}} \left(\bigvee_{i \in \{0,1,2\}} x_{i,j} \right)$$

2. (trivial) **Pour toute** ligne, **pour toute** colonne, **pour toute** bande, **il existe** une case qui satisfait à la fois une bande de la ligne et une de la colonne.

$$\bigwedge_{i \in \{0,1,2\}} \bigwedge_{j \in \{0,1,2\}} \bigwedge_{b_i^{(k)}, k \in \{1,2\}} \neg \left(\neg x_{i,j} \wedge x_{i,j} \right)$$

soit,

$$\bigwedge_{i \in \{0,1,2\}} \bigwedge_{j \in \{0,1,2\}} \bigwedge_{b_i^{(k)}, k \in \{1,2\}} \left(x_{i,j} \vee \neg x_{i,j} \right)$$

3. **Pour toute** ligne i, **pour toute** bande de taille b associée à la ligne, **il existe** *autant* de cases vraies $x_{i,j}$ que ne l'exprime la taille de chaque bande.
   Cela donne au cas par cas :
   (a) Une seule bande :
      i. Pour toute ligne i où $b = 3$,

$$\bigwedge_{j\in\{0,1,2\}} \left(\bigvee_{l=1}^{n=1} x_{i,j} \right)$$

⊤	⊤	⊤

      ii. Pour toute ligne i où $b = 2$,

$$(\neg x_{i,0} \vee \neg x_{i,2}) \wedge x_{i,1} \wedge (x_{i,0} \vee x_{i,2})$$

⊤	⊤	⊥	ou	⊥	⊤	⊤

      iii. Pour toute ligne i où $b = 1$,

$$\bigwedge_{\substack{j,j'\in\{0,1,2\} \\ j<j',j''\neq j,j'}} (\neg x_{i,j} \vee \neg x_{i,j'} \vee x_{i,j''})$$

⊤	⊥	⊥	ou	⊥	⊤	⊥	ou	⊥	⊥	⊤

      iv. Pour toute ligne i où $b = 0$,

$$\bigwedge_{j\in\{0,1,2\}} \left(\bigvee_{l=1}^{n=1} \neg x_{i,j} \right)$$

⊥	⊥	⊥

   (b) Deux bandes, pour toute ligne i :

$$\left(\bigvee_{l=1}^{n=1} x_{i,0} \right) \wedge \left(\bigvee_{l=1}^{n=1} \neg x_{i,1} \right) \wedge \left(\bigvee_{l=1}^{n=1} x_{i,2} \right)$$

⊤	⊥	⊤

   Idem pour toute colonne.

1.3 Parcours et représentation par énumération

On va parcourir la grille et énumérer les combinaisons possibles premièrement suivant les contraintes de lignes ensuite selon les contraintes de colonnes.

Par ligne (respectivement colonne), on crée si nécessaire une clause représentant les cas possibles quant aux cases dont on ne connait pas encore la nature.

Pour chaque case contenue dans une ligne i donnée :

1. Si elle contient une information (1 ou -1), on en créer une clause à part entière qui sera jointe ($\wedge$) à toutes les autres.
2. Si elle ne contient pas d'informations (0) et conditionnellement au contraintes $c_i^{(k)}, k \in \{1, 2\}$, on énumère les possibilités sous forme disjonctive.

On effectue de même par colonne et relativement aux contraintes de la colonne.

Algorithm 1 Énumération selon la valeur de(s) bande(s) de chaque lignes

for ligne $i \to n$ **do**
  for colonne $j \to 2$ **do** ▷ encodage de l'information connue
    if $x_{i,j} == \bot$ **then**
      Créer une nouvelle *clause* avec $-x_{i,j}$
    else if $x_{i,j} == \top$ **then**
      Créer une nouvelle *clause* avec $x_{i,j}$
    end if
  end for
  for colonne $j \to 2$ **do** ▷ application de la contrainte 1
    if $\nexists\, clause_i$ **then**
      Créer $clause_i$
    end if
    Ajouter $x_{i,j}$ à $clause_i$
  end for
  if nombre de *bandes* $== 1$ **then**
    if $bande_i == 0$ **then** ▷ si la taille de la bande est 0
      for colonne $j \to 2$ **do**
        Créer une nouvelle *clause* avec $\neg x_{i,j}$
      end for
    else if $bande_i == 1$ **then** ▷ si la taille de la bande est 1
      $J = \{0, 1, 2\}$
      for colonne $j'' \to 2$ **do**
        $j = min(J \setminus j'')$
        $j' = max(J \setminus j'')$
        Créer une nouvelle *clause* avec $(\neg x_{i,j} \vee \neg x_{i,j'} \vee x_{i,j''})$
      end for
    else if $bande_i == 2$ **then** ▷ si la taille de la bande est 2
      Joindre la FNC $(\neg x_{i,0} \vee \neg x_{i,2}) \wedge (x_{i,0} \vee x_{i,2}) \wedge x_{i,1}$
    else if $bande_i == 3$ **then** ▷ si la taille de la bande est 3
      for colonne $j \to 2$ **do**
        Créer une nouvelle *clause* avec $x_{i,j}$
      end for
    end if
  else if nombre de *bandes* $== 2$ **then**
    Créer une nouvelle *clause* avec $(x_{i,0} \vee \neg x_{i,1} \vee x_{i,2})$
  end if
  Joindre $(\wedge)$ toutes les *clauses*
end for

Chapitre 2

Question 2 : Les Bandes

2.1 Problème et notation

Une suite de cases noires représente une *bande*.
Une bande appartient à une ligne ou colonne et commence à une position donnée.
Comme il peut y avoir plusieurs bandes sur une même ligne ou colonne, chaque bande doit être identifiable.

2.1.1 Variables

On représente une bande par

$$LBande_{pos,ID,start}$$

où,

L|**C** précise s'il s'agit d'une ligne ou d'une colonne.

pos désigne le numéro de la ligne ou la colonne où est située la bande.

ID désigne, s'il existe plusieurs bandes sur la ligne ou colonne, la bande que l'on considère.

start désigne le numéro de colonne ou ligne à partir de laquelle commence la bande.

On défini deux fonctions donnant la longueur d'une bande :

$\texttt{L}(pos, ID)$ donnant la longueur de la ID^{eme} bande à la ligne pos.

$\texttt{C}(pos, ID)$ donnant la longueur de la ID^{eme} bande à la colonne pos.

2.1.2 Sémantique

On définit pour sémantique, dans le cas d'une ligne (raisonnement équivalent pour chaque colonne) :

$$\begin{gathered} LBande_{pos,ID,start} = \top \iff (x_{pos,j} = \top) \wedge (x_{pos,j+1} = \bot) \\ \forall j \in [start, start + \delta],\ \delta = \texttt{L}(pos, ID) \end{gathered} \tag{2.1}$$

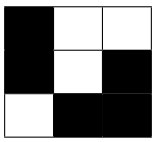


TABLE 2.1 – Grille 3×3

Par exemple, on aura pour la grille 2.1 :

$\texttt{LBande}_{3,1,2} = \top$, sachant $L(3,2) = 2$
et
$x_{3,j} = \top$, pour tout j dans l'intervalle $[3, 2 + \texttt{L}(3,2)]$
et
$x_{3,2+\texttt{L}(3,2)+1} = \bot$
où $x_{pos,j+1}$ donne $\bot$ que ce soit parce qu'il existe une case vide ou parce qu'on excède la taille de la grille.

2.2 Contraintes

Cinq contraintes ont été extraites de la problématique :

1. Une bande comporte par définition des implications sur son voisinage.
2. Au moins une des positions doit être valide.
3. Une bande ne peut avoir qu'une seule position.
4. Les bandes doivent respecter l'ordre qui est imposé par le jeu.
5. Il existe des case vides en dehors des bandes.

2.2.1 Énumération des possibles

On définit, pour chaque position potentielle d'une bande, une variable : $a, b, c, d, ...$
Chaque variable implique des conditions de véracité pour les $\texttt{L}(pos, ID)+1$ cases suivantes.

$$
\begin{aligned}
a &\implies \{X_{i,0}, X_{i,1}, X_{i,2}, \neg X_{i,3}\} \\
b &\implies \{X_{i,1}, X_{i,2}, X_{i,3}, \neg X_{i,4}\} \\
c &\implies \{X_{i,2}, X_{i,3}, X_{i,4}, \neg X_{i,5}\} \\
d &\implies \{X_{i,3}, X_{i,4}, X_{i,5}, \neg X_{i,6}\} \\
e &\implies \{X_{i,4}, X_{i,5}, X_{i,6}, \neg X_{i,7}\}
\end{aligned}
$$

Cette suite d'implication est alors concaténée à l'aide de conjonction en vue de créer la contrainte associée à la bande et la ligne.

Les implications décrites possèdent une FNC associée, on aura par exemple pour :

$$a \implies \{X_{i,0} \wedge X_{i,1} \wedge X_{i,2} \wedge \neg X_{i,3}\},$$

la FNC :

$$(\neg a \vee X_{i,0}) \wedge (\neg a \vee X_{i,1}) \wedge (\neg a \vee X_{i,2}) \wedge (\neg a \vee \neg X_{i,3})$$

2.2.2 Au moins une position

Une conséquence de l'énumération des possibles telle que décrite est qu'on ne garantit pas jusqu'à présent qu'il y ait au moins une solution.
Il faut joindre une clause qui précise qu'au moins une des positions se réalise :

$$(a \vee b \vee c \vee ...)$$

2.2.3 Une position au maximum

On a une série de positions possible dont une au moins se réalise, il faut maintenant borner le nombre de position possible, c'est-à-dire garantir qu'il ne puisse y avoir plus d'une position valide.
En effet, on autorise jusqu'à présent le cas suivant :

x_0	x_1	x_2	x_3	x_4	x_5	x_6
$\top$	$\top$	$\top$	$\bot$	$\top$	$\top$	$\top$

C'est-à-dire que l'implication a ne conditionne aucunement la véracité de l'implication e :

$$a \implies \{X_{i,0}, X_{i,1}, X_{i,2}, \neg X_{i,3}\}$$

$$e \implies \{X_{i,4}, X_{i,5}, X_{i,6}, \neg X_{i,7}\}$$

Afin de pallier à cela, on introduit de nouvelles contraintes pour éviter les cas de super-positions. Il y a plusieurs façons de les spécifier.

1. La position d'un bande implique qu'elle ne se trouve pas ailleurs, ce pour chaque bande :

   $$a \implies (\neg b \wedge \neg c \wedge \neg d \wedge ...) \wedge$$
   $$b \implies (\neg a \wedge \neg c \wedge \neg d \wedge ...) \wedge$$
   $$c \implies (\neg a \wedge \neg b \wedge \neg d \wedge ...) \wedge ...$$

   Ce qui peut se simplifier, compte tenu de la nature de la clause, en :

   $$a \implies (\neg b \wedge \neg c \wedge \neg d \wedge ...) \wedge$$
   $$b \implies (\neg c \wedge \neg d \wedge \neg e \wedge ...) \wedge$$
   $$c \implies (\neg d \wedge \neg e) \wedge ...$$

2. Aucun couple de bandes n'est possible :

   $$\neg(a \wedge b) \wedge \neg(a \wedge c) \wedge \neg(a \wedge d) \wedge ...$$

   Ce qui donne en FNC :

   $$(\neg a \vee \neg b) \wedge (\neg a \vee \neg c) \wedge (\neg a \vee \neg d) \wedge ...$$

3. La valuation d'une bande superposée à elle-même est rendue impossible par l'implication associée à chaque bande.
   Cependant, seul les cas où l'implication d'une bande ne conditionne aucunement la véracité de l'implication de la position cette même bande à un autre endroit, doit être traité :

$$a \implies \neg e$$

   dont la FNC utilisée est :

$$(\neg a \vee \neg e)$$

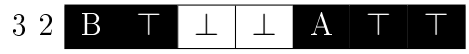
Le choix de la façon de spécifier la contrainte conduit à la question suivante : la redondance d'information accélère-t-elle la recherche de satisfaisabilité ?

2.2.4 De l'ordre des bandes

Les bandes sont spécifiées dans un ordre donné en marge de la grille.
Il y a donc une contrainte inhérente à l'ordre dans lequel sont données les bandes.

Soit l'exemple :

On sait qu'on ne peut avoir en aucun cas quelque chose de similaire à :



On joint explicitement une série de clauses qui précise, selon l'exemple :

$$A_k \equiv LBande_{i,1,k}$$
$$B_l \equiv LBande_{i,2,l}$$

et qui impose la contrainte,

$$LBande_{i,1,k} \implies \neg LBande_{i,2,l}$$

c'est-à-dire

$$A_k \implies \neg B_l$$
$$\forall l < k$$

Cette contrainte s'exprime aussi comme la conjonction des implications :

$$A_k \implies \left(\bigwedge_j^{k+1} \neg B_j \right),$$

$$B_l \implies \left(\bigwedge_j^{l+1} \neg C_j \right),$$

...

2.2.5 Case(s) vide(s) entre les bandes

Pour toute bande différente appartenant à la même ligne/colonne, il existe au moins x cases vides :

$$b - 1 \leq x \leq n - \sum_{k} \mathtt{L}(i, ID_k)$$

où,

le nombre de bandes, sur une ligne i, est $b = \sum ID$

n est la longueur de la grille

Les cases vides peuvent se trouver à différents endroits :

1. En début de ligne/colonne, c'est-à-dire avant la première bande.
2. Entre deux bandes, où il faudra calculer le nombre de cases vides entre chaque bande et la suivante.
3. En fin de ligne/colonne, c'est-à-dire après la dernière bande.
4. En combinaison des cas précédents.

Intuitivement, il s'agit de cerner les cases vides plausibles sur une ligne/colonne, par exemple :

A				B	
$\top$	$\bot$	$\bot$	$\bot$	$\top$	$\top$
$\uparrow k$	$\neg x_l$			$\uparrow j$	

Soit A et B, deux bandes commençant respectivement à la position k et j,

$$(A_k \wedge B_j) \implies (\bigwedge_{l=k+1}^{j-1} \neg x_l)$$

Ce qui se traduit en FNC par :

$$(\neg A \vee \neg B \vee \neg x_{k+1}) \wedge (\neg A \vee \neg B \vee \neg x_{k+\ldots}) \wedge (\neg A \vee \neg B \vee \neg x_{j-1})$$

Et de façon générale, dans le cas de cases entre deux bandes :

$$\bigwedge_{\forall k,j \mid k<j} \left(\bigvee_{ID} \neg ID \vee \neg x_k \right)$$

2.2.6 Contraintes implicites

De l'interdépendance des contraintes de lignes et de colonnes

La taille des bandes apparaissant dans toutes les lignes est équivalent à celle des bandes de toutes les colonnes.

$$\sum_i \sum_k \mathtt{L}(i, ID^{(k)}) = \sum_j \sum_k \mathtt{C}(j, ID^{(k)})$$

Cela se comprend aisément par le fait qu'une valuation donnée à une case vaut pour les deux dimensions (ligne et colonne)
Ce constat intervient au sujet de la nécessité ou non d'introduire une clause liant les conditions sur les colonnes et sur les lignes.
Le découpage du problème en terme de ligne et colonne incline à penser qu'il s'agit de situations distinctes qu'il faudrait corréler à un moment ou l'autre.
On se figure rapidement qu'une solution du problème, c'est-à-dire une valuation pour la grille, est la valuation pour chaque case telle qu'elle répond adéquatement à la double contrainte (ligne et colonne).

Autrement dit une valuation qui respecterait les contraintes liées à une ligne serait automatiquement invalidée si au moins une des valuations de ses cases ne répondait pas adéquatement à la contrainte de la colonne dans laquelle elle se trouve.

2.2.7 Remarques

Une bande est contenue dans la grille ou dans l'espace délimité par la bande suivante.

$$max(start_k) = \begin{cases} start_{k+1} - \mathtt{L}(pos, ID_k) - 1, & \text{si } ID > 1 \\ n - \mathtt{L}(pos, ID_k), & \text{sinon} \end{cases}$$

où,
n est la taille de la grille
ID_k est la k^{eme} bande
$k = \{0, ..., b\}$
b est le nombre de bandes sur la ligne/colonne, $b = \sum ID$
$start_k$ est la position de départ de la bande avec ID_k.

Deux bandes de la même ligne/colonne ne peuvent se superposer.

$$min(start_k) = \begin{cases} start_{k-1} + \mathtt{L}(pos, ID_{k-1}) + 1, & \text{si } ID > 1 \\ 0, & \text{sinon} \end{cases}$$

où,
ID_k est la k^{eme} bande
$k = \{0, ..., b\}$
b est le nombre de bandes sur la ligne/colonne, $b = \sum ID$
$start_k$ est la position de départ de la bande avec ID_k.

Chapitre 3

Question 3 : Résolution 3, 2

L'implémentation de la solution a directement été pensée de façon générique, afin de répondre au plus grand nombre de problèmes.
Suite à quoi, celle-ci fonctionne peu importe la taille de la grille *size* et quelque soit le nombre de bandes *tape*, pourvu que les règles du jeu soient respectées.

La **Question 3** est alors un cas particulier de la question suivante où $size = 3$ et $tape = 2$.

L'implémentation, le codage et les diverses représentations sont décrites à la **Question 4** et valent pour la question présente.

Chapitre 4

Question 4 : Résolution *size* et *tape*

4.1 Codage

Soit la caractérisation du problème au moyen des variables suivantes :

$size$ est la taille de la grille, le nombre de cases par colonne ou ligne.

pos est le numéro de la ligne/colonne.

$start$ est le numéro de la colonne/ligne.

ID est le numéro de la bande sur la ligne ou colonne.

$tapes$ est le nombre de bande possible.

$offset$ est un pointeur vers la première case après la bande précédente.

Le problème est composé de $size$ cases.
On sait que chaque case sera soit vraie ou fausse.
On sait qu'au maximum, on aura $size \times tapes$ positions possible et donc au maximum $tapes \times size$ variables par ligne/colonne.
Pour une bande ID de la ligne pos commençant à la colonne $start$, il faut aussi ajouter un certain $offset$.

L'identifiant d'une bande à une certaine position est donné par :

$$offset + (pos \times size \times tapes) + (ID \times size) + start$$

4.2 Représentation

4.2.1 Case, ligne et colonne

Case

Une case est représentée par son *adresse*, par exemple : la case à la i^{eme} ligne $(i \times size)$, à la position $j + 1$, car la ligne 1 porte l'indice 0.

De même, vis-à-vis d'une colonne : la case à la j^{eme} colonne $(j \times size)$, à la position $i+1$, car la colonne 1 porte l'indice 0

```
// Row Box
inline int ClausesGenerator::RBOX(size_t i, size_t j) const{
        return i*this->_problem->size + j + 1;
}
// Column Box
inline int CBOX(size_t j, size_t i) const{
        return i*this->_problem->size + j + 1;
}
```

Listing 4.1 – Cases

Ligne et colonne

Une colonne ou ligne est représentée comme un vecteur d'entier.

```

// Row
inline const std::vector<std::vector<int>>& ROWS() const{
        return this->_problem->rows;
}

// Column
inline const std::vector<std::vector<int>>& COLS() const{
        return this->_problem->cols;
}
```

Listing 4.2 – Ligne et colonne

4.2.2 Forme normale conjonctive

Une *clause* sera représentée par un vecteur d'entier, où chaque entier est une variable calculée telle que décrite par le codage précédent.
Une *FNC* sera représentée par un vecteur de clause.

```
typedef std::vector<int> Clause;
typedef std::vector<Clause> FNC;
```

Listing 4.3 – clause et FNC

4.2.3 Bande

Une bande d'une ligne est définie comme $LBande_{pos,ID,start}$, c'est-à-dire la variable entière :

```
//Bande
template<Which D>
inline int TAPE(size_t pos, size_t ID, size_t start) const{
        if(D == Rows) return RTAPE(pos, ID, start);
        else return CTAPE(pos, ID, start);
}

//Row Tape, LBande_{pos,ID,start}
inline int RTAPE(size_t pos, size_t ID, size_t start) const{
        return this->_Loffset + pos*this->_problem->size*this->_problem
           ->tapes + start*this->_problem->tapes + ID;
}

//Column Tape, CBande_{pos,ID,start}
inline int CTAPE(size_t pos, size_t ID, size_t start) const{
        return this->_Coffset + pos*this->_problem->size*this->_problem
           ->tapes + start*this->_problem->tapes + ID;
}
```

Listing 4.4 – Une bande comme variable

4.3 Implémentation des contraintes

4.3.1 Implication des bandes

Chaque bande implique des contraintes sur les cases voisines, par exemple :

$$a \implies \{X_{i,0} \wedge X_{i,1} \wedge X_{i,2} \wedge \neg X_{i,3}\}$$

```
template<Which D>
inline void generateClauseBoxes(size_t pos, size_t ID, size_t start){
        std::vector<int> lits;
        //boxes belonging to the tape
        for(size_t j = start; j < start + SIZE<D>(pos, ID); ++j){
                lits.push_back(BOX<D>(pos, j));
        }
        //the box that should be excluded from the tape
        if(start + SIZE<D>(pos, ID) < this->_problem->size){
                lits.push_back(-BOX<D>(pos, start + SIZE<D>(pos, ID)));
        }
        implies(TAPE<D>(pos, ID, start), lits);
}
```

Listing 4.5 – Une bande en FNC

où BOX dénote la numéro de la case :

```
template<Which D>
```

```
inline int BOX(size_t i, size_t j) const{
        if(D == Rows) return RBOX(i, j);
        else return CBOX(i, j);
}

inline int RBOX(size_t i, size_t j) const{
        return i*this->_problem->size + j + 1;
}

inline int CBOX(size_t j, size_t i) const{
        return i*this->_problem->size + j + 1;
}
```

Listing 4.6 – Case

On transforme ensuite l'implication en FNC,

$$(\neg a \vee X_{i,0}) \wedge (\neg a \vee X_{i,1}) \wedge (\neg a \vee X_{i,2}) \wedge (\neg a \vee \neg X_{i,3})$$

```
inline void implies(int conjonction, const std::vector<int>& lits){
        for (int lit : lits){
                this->_final->push_back({-conjonction, lit});
        }
}
```

Listing 4.7 – Implication en FNC

Le procédé est symétriquement identique pour les bandes appartenant aux colonnes.

4.3.2 Une position au maximum

Cette contrainte doit garantir qu'il ne puisse y avoir plus d'une position valide :

$$a \implies (\neg b \wedge \neg c \wedge \neg d \wedge ...) \wedge$$

$$b \implies (\neg c \wedge \neg d \wedge \neg e \wedge ...) \wedge$$

$$c \implies (\neg d \wedge \neg e) \wedge ...$$

Là aussi l'implication est donnée sous forme FNC (listing 4.7) :

$$(\neg a \vee \neg b) \wedge (\neg a \vee \neg c) \wedge (\neg a \vee \neg d) \wedge ...$$

```
template<Which D>
inline void generateClauseAtMostOne(size_t pos, size_t ID, size_t start)
    {
        std::vector<int> lits;
        for(size_t j = start + 1; j < this->_problem->size; ++j){
                lits.push_back(-TAPE<D>(pos, ID, j));
```

```
        }
        implies(TAPE<D>(pos, ID, start), lits);
}
```

Listing 4.8 – Au moins une position

4.3.3 Au moins une position

Il s'agit de garantir qu'il y ait au moins une position occupée par une bande :

$$(a \vee b \vee c \vee ...)$$

```
Clause atLeastOne;
for(size_t start = 0; start < this->_problem->size; ++start){
        atLeastOne.push_back(TAPE<D>(pos, ID, start));
}
this->_final->push_back(atLeastOne);
```

Listing 4.9 – Au moins une position

4.3.4 De l'ordre des bandes

Cette contrainte vise à faire respecter l'ordre des bandes :

$$\left(A_k \implies \left(\bigwedge_j^{k+1} \neg B_j\right)\right) \wedge \left(B_l \implies \left(\bigwedge_j^{l+1} \neg C_j\right)\right) \wedge \cdots$$

C'est-à-dire que, relativement à toute bande, la bande suivante ne peut précéder la bande courante.
Chaque implication est transformée en FNC et jointe aux autres.

```
// for a given tape
template<Which D>
inline void generateClauseOrder(size_t pos, size_t ID){
  for(size_t start = 0; start + SIZE<D>(pos, ID) <= this->_problem->size
      ; ++start){
    generateClauseOrder<D>(pos, ID, start);
  }
}
// the ordering constraint is imposed according the other tapes
template<Which D>
inline void generateClauseOrder(size_t pos, size_t ID, size_t start){
  std::vector<int> lits;
  for(size_t j = 0; j <= start + SIZE<D>(pos, ID); ++j){
    lits.push_back(-TAPE<D>(pos, ID+1, j));
  }
  implies(TAPE<D>(pos, ID, start), lits);
```

```
}
```

Listing 4.10 – Respect de l'ordre

4.3.5 Case(s) vide(s) entre les bandes

On crée des clauses précisant l'existence de cases vides au voisinage des bandes.
On distingue le cas où il y aurait des cases vides en début de ligne :

```
template<Which D>
inline void generateClauseHolesBegin(size_t pos){
        // first tape
        const size_t ID = 0;
        for(size_t start = 0; start + SIZE<D>(pos, ID) <= this->_problem
           ->size; ++start){
                std::vector<int> lits;
                for(size_t d = 0; d < start; ++d){
                        lits.push_back(-BOX<D>(pos, d));
                }
                implies(TAPE<D>(pos, ID, start), lits);
        }
}
```

Listing 4.11 – Cases vides (*Hole*) avant la première bande

De même, on évalue le cas où il y aurait des cases vides après la dernière bande :

```
template<Which D>
inline void generateClauseHolesEnd(size_t pos){
        // last tape
        const size_t ID = TAPES<D>().at(pos).size() - 1;
        for(size_t start = 0; start + SIZE<D>(pos, ID) <= this->_problem
           ->size; ++start){
                std::vector<int> lits;
                for(size_t d = start + SIZE<D>(pos, ID); d < this->
                   _problem->size; ++d){
                        lits.push_back(-BOX<D>(pos, d));
                }
                implies(TAPE<D>(pos, ID, start), lits);
        }
}
```

Listing 4.12 – Cases vides après la dernière bande

Les dernier cas est celui de cases vides existantes entre deux bandes ID et $ID + 1$:

```
template<Which D>
inline void generateClauseHoles(size_t pos, size_t ID, size_t start){
        for(size_t j = start + SIZE<D>(pos, ID) + 1; j < this->_problem
           ->size; ++j){
                std::vector<int> conjonction = {TAPE<D>(pos, ID, start),
                    TAPE<D>(pos, ID+1, j)};
```

```
5                   std::vector<int> lits;
6                   for(size_t d = start + SIZE<D>(pos, ID); d < j; ++d){
7                           lits.push_back(-BOX<D>(pos, d));
8                   }
9                   implies(conjonction, lits);
10          }
11  }
```

Listing 4.13 – Cases vides entre 2 bandes

Chapitre 5

Question 5 : Unicité

Afin de démontrer l'unicité d'une solution, il suffit de montrer qu'il n'en existe pas d'autre.

On commence par résoudre le problème afin d'obtenir une première solution.
On génère une contrainte à partir de la solution trouvée.
On insère et ajoute la contrainte supplémentaire au solveur et on essaie de résoudre à nouveau le problème.

Si miniSAT ne trouve pas de nouvelle solution, c'est que la première était unique.

5.1 Solution

Une solution est définie comme une grille où chaque case contient la valuation qui compose la solution.

```
typedef struct Solution{
        size_t size = 0;
        size_t tapes = 0;
        std::vector<std::vector<int>> rows;
        std::vector<std::vector<int>> cols;
        std::vector<std::vector<int>> grid;
} Solution;
```

Listing 5.1 – Solution

5.2 Test d'unicité

Le test d'unicité se déroule comme suit :
- Création d'une contrainte à partir de la solution supposée unique du problème.
- Ajout au problème de la nouvelle contrainte.
- Exécution du solveur.

– Évaluation de l'issue du solveur.

```
SolutionInhibitor inhibitor;
FNC negation;
// compute a solution and its negation as a clause
inhibitor.inhibit(solution, negation, generator);
// add clause to the solver as a new constraint
SolverTranslator::toSolver(negation, solver);
TERMINAL.print("Testing unicity ");
solver.solve();
printer.print(solver.okay());
// evaluate results of the solver
if(solver.okay()){
        TERMINAL.println("NOT UNIQUE", TERMINAL.WARNING);
        Solution solution2;
        TERMINAL.println("");
        // displays both solutions
        SolverTranslator::fromSolver(solver, problem, solution2);
        printer.print(solution2, 1);
        TERMINAL.println("");
        solutions.push_back(solution2);
}
else{
        TERMINAL.println("UNIQUE", TERMINAL.OKGREEN);
}
```

Listing 5.2 – Test d'unicité

La génération d'une nouvelle contrainte à partir de la première solution trouvée se fait en niant la valuation de toutes ses cases.

```
void inhibit(const Solution& solution, FNC& constraints, const
   ClausesGenerator& generator){
        Clause clause;
        for(size_t i = 0; i < solution.grid.size(); ++i){
                for(size_t j = 0; j < solution.grid.at(i).size(); ++j){
                        // negation boxes
                        if(solution.grid.at(i).at(j) == 1){
                                clause.push_back(-generator.RBOX(i,j));
                        }else if (solution.grid.at(i).at(j) == -1){
                                clause.push_back(generator.RBOX(i,j));
                        }
                }
        }
        // generates a new constraint
        constraints.push_back(clause);
}
```

Listing 5.3 – Création de la contrainte

Chapitre 6

Question 6 : Taille Variable

6.1 Généralisation

Au sujet de la résolution d'instances particulières du jeu *Pattern* pour lesquelles la taille serait variable :

L'application proposée commence par analyser le fichier contenant l'énoncé du problème au moyen d'un *parser*.
Le fichier spécifiant les paramètres particuliers du problème, l'application est en mesure de fonctionner avec n'importe quelle taille de grille ou de bande.

L'implémentation décrite au Chapitre 4 a été réalisée de façon à fonctionner avec des paramètres variables.

6.2 Complexité

L'analyse des contraintes, telles que décrites dans les chapitres précédents, permettent de statuer sur la complexité de la formulation des contraintes en vue d'évaluer la satisfaisabilité du problème.

Dans le pire des cas :

1. L'énumération des positions se fait en $O(n^2)$
2. Contraindre à au moins une position est obtenu en $O(n)$
3. Contraindre à n'avoir au maximum qu'une position se fait en $O(n^2)$
4. Garantir l'ordre des bandes est réalisé en $O(n^2)$
5. Imposer le nombre de cases vides est atteint en $O(n^2)$

La création de clauses pour une bande appelle $n\times$les contraintes, donc la génération des toutes les clauses pour les n bandes se fait en $O(n^4)$

Chapitre 7

Question 7 : Génération

Afin de générer des grilles, comme instance du problème SAT, pour lesquelles il existerait une unique solution, il faut rendre les *entrées* variables.
Les *entrées* sont les paramètres du jeu, c'est-à-dire le **nombre** et **la taille des bandes** et **la taille du problème** (de la grille).

L'idée serait de faire des littéraux à partir de chaque paramètre possible.

On pourrait définir de nouvelles variables représentant la taille des bandes.
C'est-à-dire que $C(pos, ID, k)$, la bande à la position pos de taille k, peut être vu comme une variable pour laquelle il sera possible d'inférer des implications, par exemple :
Si la bande 1 est de taille 1 alors ça implique que...
Si la bande 1 est de taille 2 alors ça implique que...

De même on pourrait définir comme variable le nombre de bande par ligne ou colonne.

Contrairement aux deux paramètres précédents qui sont limités, la taille du problème pourrait être potentiellement infinie, elle devra donc être nécessairement limitée.

À partir de ces variables/littéraux, il sera alors possible décrire les clauses dont la conjonction définirait une valuation pour la taille des bandes, le nombre de bandes, dans un problème de taille donné, c'est-à-dire une grille.

Ensuite, afin d'obtenir une nouvelle grille, il suffira de passer au solveur la précédente grille sous forme de contrainte, celui-ci donnera alors une nouvelle valuation pour toutes les variables, soit une nouvelle grille.

Chapitre 8

Usage

8.1 Compilation

Pour compiler, lancer dans un terminal à partir du répertoire du projet :

```
make rebuild
```

8.2 Execution

Pour exécuter une instance du programme, introduire par exemple :

```
./run grid/2 sol/2 -tvu
```

où

`grid/2` est le chemin vers l'énoncé 2
`sol/2` est le chemin du fichier de sortie (optionnel)
`-tvu` sont des options

Liste des options :

1. `-u` : Lancer le test d'unicité
2. `-v` : mode *verbose*
3. `-t` : afficher le temps
4. `-h --help` : rappel de l'aide
5. `--nocolor` : ne pas afficher les couleurs

Les options peuvent être ajoutées dans n'importe quel ordre.