UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE DE LA MAÎTRISE
EN MATHÉMATIQUES ET INFORMATIQUE APPLIQUÉES

PAR
HUGO CLOUTIER

ÉNUMÉRATION DE POLYOMINOS À DEUX ET À TROIS DIMENSIONS

DÉCEMBRE 2010

ÉNUMÉRATION DE POLYOMINOS À DEUX ET À TROIS DIMENSIONS

Hugo Cloutier

SOMMAIRE

Les polyominos sont étudiés depuis environ 60 ans. Au fil du temps, plusieurs classes de polyominos ont été définies par des chercheurs et dans ce mémoire, une nouvelle classe de polyominos sera étudiée : les polyominos minimaux. Dans ce mémoire, nous voyons les polyominos convexes, les polyominos et les polyominos minimaux. Le but est d'énumérer ces polyominos pour chacun de ces ensembles en utilisant différents moyens comme des algorithmes, des récurrences, des séries génératrices et des formules directes.

Des définitions de base sont présentées au chapitre 1. Elles précisent ce que sont les polyominos étudiés dans ce mémoire. Au chapitre 2, les polyominos convexes sont énumérés à l'aide d'une série génératrice. Cette façon d'énumérer les polyominos convexes existait déjà dans un article de M. Bousquet-Mélou [2] et ailleurs. Au chapitre 3, une méthode récursive d'énumération et de construction des polyominos où ceux-ci sont construits colonne par colonne est présentée. Cette méthode n'a jamais été publiée, car son efficacité reste à déterminer. Le chapitre 4 porte sur l'énumération de polyominos d'aire minimale, une nouvelle classe de polyominos. Les polyominos de cet ensemble sont inscrits dans un rectangle. C'est-à-dire qu'ils touchent chaque côté d'un rectangle. Ils sont énumérés à l'aide d'une récurrence, d'une formule directe et d'une série génératrice. De plus, une généralisation des ces polyominos minimaux se trouve au chapitre 5. Ils s'appellent alors polyominos de volume minimal et ils sont énumérés par une série génératrice. Cette série génératrice donne sous certaines restrictions une formule directe et une récurrence linéaire à coefficients polynomiaux.

L'idée principale motivant ce travail est qu'on peut utiliser un algorithme énumérant les polyominos généraux d'une aire donnée et améliorer cet algorithme en considérant que ces polyominos sont inscrits dans différents rectangles. Alors, pour certains rectangles, les polyominos considérés sont minimaux et peuvent être énumérés par une formule directe.

ENUMERATION OF TWO AND THREE DIMENSIONS POLYOMINOES

Hugo Cloutier

ABSTRACT

In this work, different sets of polyominoes are studied and enumerated in different ways. We find in the following pages enumerations of convex polyominoes, polyominoes and minimal polyominoes. Convex polyominoes are enumerated by using recursive formulas and generating functions. An algorithm is used to enumerate polyominoes. Minimal polyominoes are enumerated by using recursive formulas, generating functions and exact formulas.

In chapter 1, we give definitions of the polyominoes studied in this work. In chapter 2, convex polyominoes are enumerated with the method of generating functions. This has already been done by M. Bousquet-Mélou [2] and others. Chapter 3 presents a way to enumerate general polyominoes by using a recursive algorithm. In this approach, polyominoes are constructed one column at the time and at each step, the number of remaining cells is reduced. This algorithm has not been published yet because its efficiency is still unknown. Chapter 4 is about the enumeration of minimal area polyominoes inscribed in a rectangle. This is a new approach. These polyominoes are enumerated by using a recursive formula, a generating function and a direct formula. In chapter 5, a generalization of minimal polyominoes is presented. Minimal area polyominoes become minimal volume polyominoes inscribed in a rectangular prism. These 3D polyominoes are enumerated by the method of generating functions and for particular cases, we have produced a direct formula and a recursive linear formula with polynomial coefficients.

The main idea behind this work is that an algorithm enumerating general polyominoes can be improved by inscribing these polyominoes in rectangles. Then for some rectangles, we have minimal area polyominoes that can be enumerated by an exact formula. Furthermore, to enhance this algorithm, it is possible to search formulas for polyominoes having a fixed number of cells in excess of the minimal polyominoes.

REMERCIEMENTS

Je souhaite remercier sincèrement les personnes et l'organisme suivants :

- M. Alain Goupil pour m'avoir transmis l'essentiel de ce dont j'avais besoin pour réaliser ce projet et pour avoir stimulé nos réflexions ;
- M. Alain Chalifour qui a initié les recherches pour ce projet en présentant quelques images ;
- M. Michel Volle pour son appui financier ;
- Tout ceux et celles qui ont participé aux réflexions hebdomadaires sur le sujet ;
- Le CLUMEQ pour la mise à notre disposition d'ordinateurs puissants.

Merci.

TABLE DES MATIÈRES

Page

SOMMAIRE i
ABSTRACT ii
REMERCIEMENTS iii
TABLE DES MATIÈRES iv
LISTE DES TABLEAUX v
LISTE DES FIGURES vi
INTRODUCTION 1
CHAPITRE 1 DÉFINITIONS 3
1.1 Polyominos aux cellules carrées 3
1.2 Polyominos tridimensionnels 4
CHAPITRE 2 ÉNUMÉRATION DE POLYOMINOS CONVEXES 7
2.1 Définitions 7
2.2 Une récurrence 9
2.3 Une série génératrice 13
CHAPITRE 3 UNE MÉTHODE PAR RÉCURRENCE 21
3.1 Introduction 21
3.2 Définitions 22
3.3 Les précolonnes 26
3.4 Opérations sur des colonnes 31
3.5 Récurrence sur l'aire des polyominos 46
CHAPITRE 4 LES POLYOMINOS D'AIRE MINIMALE 49
4.1 Introduction 49
4.2 Définitions 49
4.3 Une description de l'ensemble des polyominos d'aire minimale 51
4.4 Une récurrence 51
4.5 Une formule directe 53
4.6 Une série génératrice 54
CHAPITRE 5 LES POLYOMINOS DE VOLUME MINIMAL 58
5.1 Introduction 58
5.2 Définitions 58
5.3 Les polyominos diagonaux 59
5.4 Polyominos non diagonaux 65
5.5 Résultat principal 70
5.6 Des formules provenant de la série génératrice des polyominos de volume minimal 75
CONCLUSION 77
ANNEXES
1 : Programme énumérant les polyominos à forme fixée 78
2 : Une formule itérative 124

3 : Un programme utilisant des filtres 137
BIBLIOGRAPHIE 166

LISTE DES TABLEAUX

Page

Tableau I Nombre de polyominos d'aire minimale dans un rectangle 57
Tableau II Nombre de polyominos d'aire minimale n 57

Tableau III Nombre de polyominos de volume minimal n 75

Tableau IV Nombre de polyominos de volume minimal dans un prisme 124

LISTE DES FIGURES

Page

Figure 1 Une cellule . . . 3
Figure 2 Deux cellules reliées par un chemin . . . 3
Figure 3 Un polyomino . . . 4
Figure 4 Un polyomino d'aire minimale inscrit dans un rectangle . . . 4
Figure 5 Une cellule tridimensionnelle . . . 5
Figure 6 Deux cellules reliées par un chemin . . . 5
Figure 7 Un polyomino tridimensionnel . . . 6
Figure 8 Un polyomino de volume minimal . . . 6

Figure 9 Un polyomino tas . . . 7
Figure 10 Un polyomino parallélogramme . . . 8
Figure 11 Un polyomino convexe . . . 8
Figure 12 Un polyomino dirigé convexe . . . 9
Figure 13 Un polyomino convexe dont le parallélogramme est ascendant . . . 9
Figure 14 Construction d'un polyomino tas . . . 15
Figure 15 Construction d'un ensemble de polyominos dirigés convexes . . . 16
Figure 16 Construction d'un ensemble de polyominos convexes . . . 19

Figure 17 Construction de polyominos colonne par colonne . . . 22
Figure 18 Colonne élémentaire simple . . . 23
Figure 19 Une colonne élémentaire . . . 24
Figure 20 Position relative de colonnes élémentaires . . . 24
Figure 21 Une colonne . . . 25
Figure 22 Une précolonne élémentaire . . . 27
Figure 23 Des colonnes élémentaires simples voisines de chaque type . . . 37
Figure 24 Une fin de polyomino en partant d'une colonne . . . 47
Figure 25 Les fins possibles . . . 47

Figure 26 Une équerre . . . 50
Figure 27 Un escalier . . . 50
Figure 28 Un polyomino croix . . . 50

Figure 29 Un polyomino de volume minimal . . . 59
Figure 30 Une équerre 3D . . . 60
Figure 31 Un polyomino de volume minimal construit sur 2 diagonales . . . 63
Figure 32 Un polyomino de volume minimal construit sur 4 diagonales . . . 63
Figure 33 Un polyomino 2DX2D . . . 65
Figure 34 Des croix gauches de types A (à gauche) et B (à droite) . . . 68
Figure 35 Positions possibles des cellules de contact . . . 72

Figure 36 Le « dépliage » d'une partie de polyomino 3D vu du haut 126
Figure 37 Positions des cellules en contact avec trois cellules 130

INTRODUCTION

Les polyominos ont été introduits par S. Golomb [6] en 1952. Depuis ce temps, le problème d'énumération de ceux-ci est resté ouvert. Toutefois, pour certains sous-ensembles de polyominos, on a trouvé des formules exactes et des séries génératrices permettant d'en faire l'énumération. En particulier, M. Bousquet-Mélou [2] a déterminé une série génératrice pour l'énumération des polyominos convexes et a apporté un premier résultat exact à propos de l'énumération de ces polyominos. Dans ce mémoire, il y a des séries génératrices et des formules exactes permettant d'énumérer les polyominos minimaux. Aussi, des algorithmes énumérant les polyominos les plus généraux existent. Actuellement, l'algorithme développé par I. Jensen [9] est le plus efficace pour faire cela. Le travail de ce mémoire a pour but de développer des méthodes d'énumération qui pourraient éventuellement conduire à des méthodes plus performante que cet algorithme.

Dans cet ouvrage, nous verrons d'abord l'énumération de polyominos convexes au chapitre 2. Pour ce faire, nous utiliserons des séries génératrices comme A. Robitaille [11] l'a fait. Au chapitre 3, Nous verrons un algorithme basé sur une récurrence qui traite les polyominos les plus généraux dont les cellules sont carrées. Ensuite, les polyominos d'aire minimale inscrits dans un rectangle seront abordés au chapitre 4 et nous généraliserons ces polyominos à ceux de volume minimal inscrits dans un prisme rectangulaire au chapitre 5. Les cellules de ces derniers sont des cubes unités. Pour énumérer ces polyominos, nous utiliserons des récurrences, des séries génératrices et des formules exactes. Ces séries génératrices seront construites selon la structure générale de ces polyominos.

Certaines solutions présentées dans ce mémoire ne semblent pas avoir été publiées ailleurs et les problèmes d'énumération de polyominos minimaux sont « nouveaux ». J'ai développé la méthode présentée au chapitre 3. Alain Goupil et Alain Chalifour

m'ont aidé à la valider. Aussi, ce qui concerne les polyominos d'aire minimale dans ce mémoire a été publié par les auteurs A. Goupil, H. Cloutier et F. Nouboud [7]. Le travail sur les polyominos de volume minimal fait par A. Goupil et H. Cloutier [8] est aussi inédit.

L'énumération de polyominos minimaux est un point de départ pour compter les polyominos généraux. En effet, nous pouvons tenter d'énumérer les polyominos minimaux auxquels des cellules sont ajoutées. La méthode du chapitre 3 peut être améliorée en l'appliquant à des polyominos d'une aire donnée et inscrits dans des rectangles parce que pour les rectangles où les polyominos sont d'aire minimale, il est possible d'utiliser les résultats du chapitre 4.

CHAPITRE 1

DÉFINITIONS

1.1 Polyominos aux cellules carrées

Définition 1.1.1 Une ***cellule*** est un carré unité orienté comme à la figure 1. Aussi, les cellules seront placées dans le plan cartésien de manière à ce que leurs centres soient à coordonnées entières et leurs côtés parallèles ou perpendiculaires à l'abscisse. On dira que ces coordonnées sont celles de ces cellules.

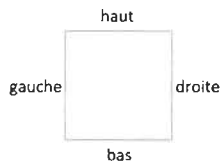

Figure 1 Une cellule

Définition 1.1.2 Un ***chemin*** relie deux cellules de coordonnées (a_1, b_1) et (a_n, b_n) s'il existe une suite de coordonnées $S = \{(a_1, b_1), (a_2, b_2), ..., (a_n, b_n)\}$ telle que ces coordonnées sont celles de cellules et $\forall (a_i, b_i) \in S - (a_n, b_n)$, $\sqrt{(a_{i+1} - a_i)^2 + (b_{i+1} - b_i)^2} = 1$.

Exemple 1.1.1 Sur la figure 2, les cellules plus sombres sont reliées par un chemin.

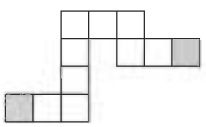

Figure 2 Deux cellules reliées par un chemin

Définition 1.1.3 Un ***polyomino*** est un ensemble de cellules tel que toute paire de cellules de cet ensemble est reliée par un chemin. Alors, on dit que cet ensemble est *4-connexe* et que ces cellules sont *4-connectées*.

Exemple 1.1.2 La figure 3 présente un polyomino.

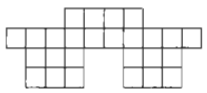

Figure 3 Un polyomino

Définition 1.1.4 Soit un rectangle de dimension $b \times h$. Alors, un polyomino ayant exactement $b + h - 1$ cellules et étant en contact avec les quatre côtés de ce rectangle est un ***polyomino d'aire minimale inscrit dans ce rectangle***.

Au chapitre 4, on énumérera les polyominos d'aire minimale inscrits dans un rectangle de dimension $b \times h$. La figure 4 illustre un tel polyomino.

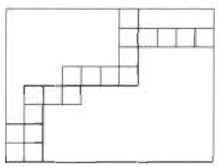

Figure 4 Un polyomino d'aire minimale inscrit dans un rectangle

1.2 Polyominos tridimensionnels

Définition 1.2.1 Dans un contexte tridimensionnel, une ***cellule*** est un cube unité qui sera placé dans l'espace tridimensionnel de manière à ce que son centre soit à coordonnées entières et ses faces soient parallèles ou perpendiculaires aux plans de coordonnées. On dira que ces coordonnées sont celles de cette cellule. Les plans de coordonnées seront nommés XOY, XOZ et YOZ.

Définition 1.2.2 Dans un contexte tridimensionnel, un ***chemin*** relie deux cellules de coordonnées (a_1, b_1, c_1) et (a_n, b_n, c_n) s'il existe une suite de coordonnées $S = \{(a_1, b_1, b_1), (a_2, b_2, c_2), ..., (a_n, b_n, c_n)\}$ telle que ces coordonnées sont celles de cellules et $\forall (a_i, b_i, c_i) \in S$ avec $1 \leq i < n$, $\sqrt{(a_{i+1} - a_i)^2 + (b_{i+1} - b_i)^2 + (c_{i+1} - c_i)^2} = 1$.

Figure 5 Une cellule tridimensionnelle

Exemple 1.2.1 Sur la figure 6, les cellules plus sombres sont reliées par un chemin.

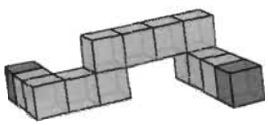

Figure 6 Deux cellules reliées par un chemin

Définition 1.2.3 Dans un contexte tridimensionnel, un ***polyomino*** est un ensemble de cellules tel que pour toute paire de cellules de cet ensemble, ces cellules sont reliées par un chemin. Alors, on dit que cet ensemble est *6-connexe* et que ces cellules sont *6-connectées*.

Définition 1.2.4 Un ***polyomino de volume minimal inscrit dans un prisme*** est composé d'exactement $b + h + k - 2$ cellules si le prisme dans lequel ce polyomino est inscrit est de dimension $b \times h \times k$. De plus, un tel polyomino est en contact avec les six faces de ce prisme.

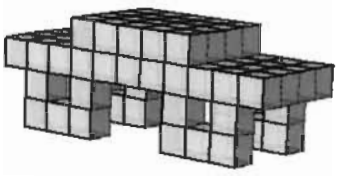

Figure 7 Un polyomino tridimensionnel

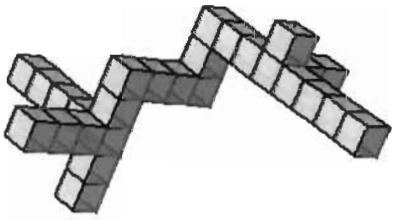

Figure 8 Un polyomino de volume minimal

La figure 8 illustre un polyomino de volume minimal. Les polyominos de volume minimal sont énumérés par des séries génératrices au chapitre 5.

CHAPITRE 2

ÉNUMÉRATION DE POLYOMINOS CONVEXES

2.1 Définitions

Les polyominos convexes sont de la forme $tas \cdot parallélogramme \cdot tas$ où le polyomino parallélogramme peut être ascendant, descendant ou dégénéré. Cette structure ressemble en partie à celle utilisée dans le mémoire d'Ariane Robitaille [11]. La différence dans la présente approche est que le cas où le polyomino parallélogramme est dégénéré est isolé. Voyons d'abord quelques définitions.

Définition 2.1.1 Dans le plan, un polyomino ***tas*** a les hauteurs de colonnes en ordre décroissant de gauche à droite. De plus, dans un tel polyomino, une colonne à droite d'une autre ne peut avoir une cellule plus basse que la plus basse cellule de cette autre colonne ni une cellule plus haute que la plus haute cellule de cette autre colonne. Le résultat d'une réflexion selon un axe vertical quelconque appliquée à un polyomino tas est un polyomino tas de droite à gauche.

Exemple 2.1.1 La figure suivante présente un polyomino tas.

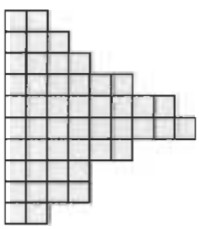

Figure 9 Un polyomino tas

Définition 2.1.2 Un polyomino est ***parallélogramme*** si chacune de ses colonnes à droite d'une autre a toutes ses cellules plus hautes (respectivement plus basse) ou à la

même hauteur que la cellule la plus basse (respectivement la plus haute) de cette autre colonne. De plus, cette colonne à droite doit avoir une cellule à la même hauteur que la cellule la plus haute (respectivement la plus basse) de cette autre colonne et l'ensemble des cellules de toute colonne doit être connexe.

Exemple 2.1.2 La figure suivante présente un polyomino parallélogramme.

Figure 10 Un polyomino parallélogramme

Définition 2.1.3 Un polyomino est ***convexe*** si tout ensemble de cellules prises dans une seule et même colonne ou sur une seule et même ligne est connexe.

Exemple 2.1.3 La figure suivante présente un polyomino convexe.

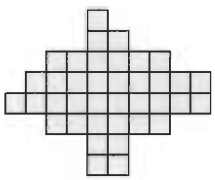

Figure 11 Un polyomino convexe

Définition 2.1.4 Un polyomino ***dirigé convexe*** est un polyomino convexe tel qu'à partir de sa cellule la plus basse de sa colonne la plus à gauche, chacune de ses cellules peut être atteinte par une suite de pas vers le haut ou vers la droite.

Exemple 2.1.4 La figure suivante présente un polyomino dirigé convexe.

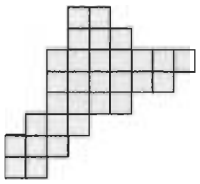

Figure 12 Un polyomino dirigé convexe

2.2 Une récurrence

Les résultats de cette section n'ont pas été publiés ailleurs. Il s'agit essentiellement de récurrences obtenues facilement en construisant les polyominos convexes de gauche à droite, colonne par colonne.

Nous supposerons dans un premier temps que les polyominos parallélogrammes sont ascendants de gauche à droite comme sur l'illustration suivante. Aussi, pour marquer la séparation dans un polyomino convexe entre la fin du parallélogramme et le début du tas de droite, dans le tas de droite, la plus grande colonne aura toujours au moins une cellule plus haute que la cellule la plus haute de la colonne suivante à droite. De manière similaire, dans la première colonne du parallélogramme, il y aura toujours une cellule strictement plus basse que la plus basse cellule de la colonne suivante.

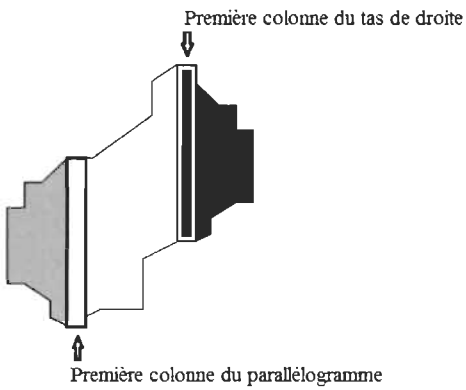

Figure 13 Un polyomino convexe dont le parallélogramme est ascendant

D'abord, voyons une récurrence pour les polyominos tas qui sont la partie en haut à droite des polyominos convexes. À cette étape de la construction, pour qu'on soit forcé de ne placer que des polyominos tas, il faut que la deuxième colonne de ce polyomino tas n'ait pas de cellule vis-à-vis la cellule la plus haute de la colonne qui la précède. De plus, dans cette deuxième colonne, il ne peut y avoir de cellule plus basse que la cellule la plus basse de la colonne précédente, car c'est un polyomino tas.

Pour tous les polyominos considérés, on notera m la hauteur de la première colonne (en ordonnant les colonnes de gauche à droite), n le nombre de cellules du polyomino et i la hauteur de la deuxième colonne.

Un polyomino tas est une colonne collée à un polyomino tas à droite et la deuxième colonne d'un polyomino tas peut être placée à $m - i + 1$ endroits et sa hauteur i peut varier de 1 à m. Selon ces observations, si $Tfin(m, n)$ est le nombre de polyominos tas d'aire n dont la première colonne a m cellules, alors nous pouvons énumérer les polyominos tas par la récurrence

$$Tfin(m,n) = \begin{cases} 0 & \text{si } n < 0 \\ 0 & \text{si } m > n \\ 1 & \text{si } n = m \\ \sum_{i=1}^{m} Tfin(i, n-m)(m-i+1) & \text{sinon.} \end{cases}$$

Dans un polyomino convexe, pour le polyomino tas de droite dont la première colonne est de hauteur m, une fois que ces m cellules sont placées, à partir de la deuxième colonne, il y a un polyominos tas qui peut être à $m - i$ endroits (la première colonne doit excéder la deuxième). De plus, la hauteur de la deuxième colonne doit être plus petite que m. Alors les polyominos tas de droite dans un polyomino convexe

sont énumérés par

$$T(m,n) = \sum_{i=1}^{m-1} Tfin(i, n-m)(m-i).$$

Dans ce qui suivra, on procédera par récurrence comme avec les polyominos tas.

Pour qu'on soit forcé de construire un polyomino parallélogramme possiblement suivi d'un polyomino tas, à la deuxième colonne du polyomino parallélogramme, il faut qu'il n'y ait pas de cellule vis-à-vis la cellule la plus basse de la colonne précédente et il doit y avoir une cellule vis-à-vis la plus haute cellule de la colonne précédente. Aussi, dans cette deuxième colonne du polyomino parallélogramme, il y a possiblement des cellules plus hautes que la cellule la plus haute de la colonne précédente (toujours selon un ordre de gauche à droite). Une fois que cette deuxième colonne est placée à l'un des $min(i, m-1)$ endroits possibles, soit il ne reste plus de cellules, soit on force le reste de la construction à être un polyomino tas, soit on poursuit avec un parallélogramme possiblement suivi d'un polyomino tas. En faisant des raisonnements semblables à ceux fait pour les polyominos tas, on trouve que la récurrence énumérant les polyominos forcés d'être un parallélogramme possiblement suivi d'un polyomino tas est

$$Par(m,n) = \sum_{i=1}^{n-m} ParFin(i, n-m) \cdot min(i, m-1)$$

où

$$ParFin(m,n) = \begin{cases} 0 & \text{si } n < 0 \\ 1 & \text{si } n = m \\ T(m,n) + \sum_{i=1}^{n-m} ParFin(i, n-m) \cdot min[i,m] & \text{sinon.} \end{cases}$$

Enfin, soit $P(m,n)$, le nombre de polyominos convexes où le parallélogramme est orienté vers le haut. Si on n'est pas forcé de placer un parallélogramme possiblement

suivi d'un tas ou simplement un tas, alors la deuxième colonne est plus grande ou égale à celle qui la précède et les constructions faites à partir de cette deuxième colonne sont énumérées par $P(i, n-m)$. Ainsi, toujours par les mêmes raisonnements que ceux faits avec les polyominos tas, la récurrence énumérant les convexes où le parallélogramme est orienté vers le haut est

$$P(m,n) = \begin{cases} 0 & \text{si } m > n \\ 1 & \text{si } m = n \\ T(m,n) + Par(m,n) + \sum_{i=m}^{n-m} P(i, n-m)(i-m+1) & \text{sinon.} \end{cases}$$

Comme le parallélogramme dans un polyomino convexe est soit vers le haut, soit vers le bas, soit les deux (dans ce dernier cas, le polyomino convexe est constitué de deux tas), le nombre total de polyominos convexes d'aire n dont la première colonne est de hauteur m est

$$C(m,n) = 2 \cdot P(m,n) - TT(m,n)$$

où $TT(m,n)$ est le nombre de polyominos convexes dont le parallélogramme est à la fois orienté vers le haut et vers le bas. On trouve la récurrence

$$TT(m,n) = \begin{cases} 1 & \text{si } m = n \\ \sum_{i=m}^{n-m} TT(i, n-m)(i-m+1) & \\ \quad + \sum_{j=1}^{m-1} Tfin(j, n-m) + T(m,n) & \text{sinon.} \end{cases}$$

Il ne reste qu'a considérer toutes les hauteurs possibles de la première colonne pour trouver $Pc(n)$, le nombre de polyominos convexes d'aire n. En effet,

$$Pc(n) = \sum_{m=1}^{n} \left(2 \cdot P(m,n) - TT(m,n)\right).$$

2.3 Une série génératrice

On peut traduire la structure d'objets combinatoires par des équations mettant en relation des séries génératrices. Ces équations sont appelées équations fonctionnelles. En général, dans ce qui suit, pour une série génératrice $S(x,y)$, il est écrit $S(x)$ pour nommer $S(x,y)$ quand le deuxième paramètre est toujours égal à y. Par exemple, $S(xy)$ représente la série à deux variables $S(x,y)$ dans laquelle x est remplacé par xy. Pour résoudre les équations fonctionnelles provenant de la structure des polyominos convexes, nous aurons besoin du lemme suivant.

Lemme 2.3.1 *Pour une équation fonctionnelle de la forme*

$$X(x) = e(x) + f(x)X(1) + g(x)X(xy) + h(x)\frac{\partial X}{\partial x}(1)$$

où $e(x), f(x), g(x)$ et $h(x)$ sont des séries formelles connues, nous avons

$$X(x) = E(x) + \frac{F(x)\left[E(1) + H(1)E'(1) - E(1)H'(1)\right] + H(x)\left[E'(1) + E(1)F'(1) - F(1)E'(1)\right]}{1 - F(1) - H'(1) - H(1)F'(1) + F(1)H'(1)}$$

où

$$E(x) = \sum_{n\geq 0} g(x)g(xy)...g(xy^{n-1})e(xy^n),$$
$$F(x) = \sum_{n\geq 0} g(x)g(xy)...g(xy^{n-1})f(xy^n)$$

et

$$H(x) = \sum_{n\geq 0} g(x)g(xy)...g(xy^{n-1})h(xy^n).$$

Une preuve de ce lemme est présentée dans le mémoire d'Ariane Robitaille [11] à la page 14.

Les séries génératrices qui suivront seront de la forme

$$S(x,y) = \sum_{m,n\geq 0} s(m,n)x^m y^n$$

où $s(m,n)$ sera le nombre de polyominos associés à cette série ayant exactement m cellules dans la colonne de gauche et n cellules en tout.

On trouve une autre version du théorème suivant dans le mémoire d'Ariane Robitaille [11].

Théorème 2.3.1 *Soit $T(x,y) = \sum_{m,n\geq 0} t(m,n)x^m y^n$, la série génératrice des polyominos tas. Alors, l'équation fonctionnelle associée aux polyominos tas est*

$$T(x) = \frac{xy}{1-xy} + \frac{1}{(1-xy)^2}T(xy).$$

De plus, la solution à cette équation fonctionnelle est

$$T(x,y) = \sum_{n\geq 0} \frac{xy^{n+1}}{1-xy^{n+1}} \prod_{i=1}^{n} \frac{1}{(1-xy^i)^2}.$$

Preuve Soit un polyomino tas possède une seule colonne, soit c'est un polyomino tas auquel on a ajouté une colonne à gauche de manière à former un polyomino tas dont la colonne de gauche est plus grande ou égale aux autres colonnes (voyez la figure 14). Dans le premier cas, on a la série $\frac{xy}{1-xy}$ et dans le second cas, pour ajouter une colonne, on double d'abord la plus grande colonne du polyomino avant l'ajout(La série génératrice de ces polyominos est alors $T(xy)$). Pour terminer la construction de la

colonne ajoutée, on place des bouts de colonne d'une longueur arbitraire (possiblement nulle) au dessus et en dessous de la colonne ajoutée comme à la figure 14.

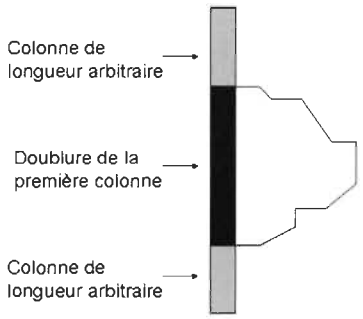

Figure 14 Construction d'un polyomino tas

Pour isoler $T(x, y)$, il suffit d'appliquer le lemme 2.3.1.

■

Un polyomino convexe peut aussi être un polyomino dirigé convexe. Alors, nous aurons besoin du théorème suivant qui est une adaptation d'un résultat présenté dans le mémoire d'Ariane Robitaille [11].

Théorème 2.3.2 *Soit* $D(x, y) = \sum_{m,n\geq 0} d(m, n)x^m y^n$, *la série génératrice des polyominos dirigés convexes. Alors, l'équation fonctionnelle associée aux polyominos dirigés convexes est*

$$D(x) = xy + xyT(x) + \frac{xy}{(1-xy)^2}\left(D(1) - D(xy)\right).$$

Preuve Dans un polyomino dirigé convexe, soit il n'y a qu'une cellule, soit il n'y a qu'une colonne de plus d'une cellule, soit le haut de la première colonne est plus haut que celui de la seconde (et cette seconde colonne a au moins une cellule), soit le haut de la première colonne est plus bas ou à la même hauteur que le haut de la seconde colonne (toujours selon un ordre de gauche à droite).

Dans le premier cas, la série associée est xy.

Pour les deuxième et troisième cas, la série associée est $xyT(x)$. En effet, dans ce cas, ce qui suit la première colonne est forcément un polyomino tas, sinon il n'y a aucune cellule à droite de cette première colonne. Alors, la première colonne peut être vue comme la première colonne d'un polyomino tas avec une cellule supplémentaire en haut.

Pour ce qui est du quatrième cas, voyez la figure 15. On construit d'abord tous les polyominos dirigés convexes ayant le haut de la première colonne à la même hauteur ou plus bas que le haut de la seconde colonne, mais, dans certains cas pour ces premières constructions, le haut de la première colonne dépasse la seconde. Ces premières constructions sont constituées d'un polyomino dirigé convexe auquel on ajoute une cellule (en gris foncé) puis deux parties de colonne (en gris pâle). La série associée à cette étape est $\frac{xy}{(1-xy)^2}D(1)$. (Lorsqu'on considère $D(1)$ cela représente les polyominos dirigés convexes de toute hauteur.) De manière similaire, la série associée à ce qu'on doit enlever des premières constructions est $-\frac{xy}{(1-xy)^2}D(xy)$.

■

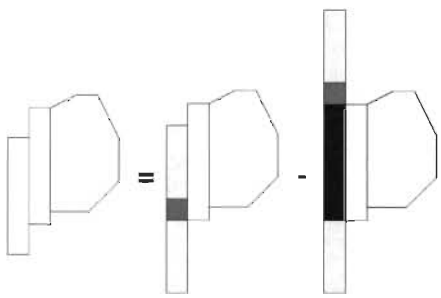

Figure 15 Construction d'un ensemble de polyominos dirigés convexes

On peut résoudre l'équation fonctionnelle du théorème 2.3.2 en appliquant le lemme 2.3.1.

Nous considérerons $P_\uparrow$ l'ensemble des polyominos convexes dont la partie qui est un polyomino parallélogramme est ascendante (voir la figure 13) et $P_\downarrow$ l'ensemble de ceux dont le parallélogramme est descendant. Par le principe d'inclusion exclusion, nous obtiendrons l'ensemble des polyominos convexes $P_c = P_\uparrow \cup P_\downarrow - P_\uparrow \cap P_\downarrow$. Remarquez que $|P_\uparrow| = |P_\downarrow|$. Au chapitre 4, cette même idée est exploitée pour énumérer les polyominos d'aire minimale. C'est cette idée qui fait que mon travail dans ce chapitre diffère de ce qu'Ariane Robitaille [11] a fait concernant la série génératrice des polyominos convexes. Voyons maintenant l'équation fonctionnelle associée à $P_\uparrow$. Cette équation fonctionnelle peut être résolue à l'aide du lemme 2.3.1 et le théorème qui suit est une adaptation d'un résultat présenté dans le mémoire d'Ariane Robitaille.

Théorème 2.3.3 *Soit $P_\uparrow(x,y) = \sum_{m,n\geq 0} p_\uparrow(m,n)x^m y^n$, la série génératrice des polyominos convexes dont la partie qui est un polyomino parallélogramme est ascendante. Alors, l'équation fonctionnelle associée à ces polyominos est*

$$P_\uparrow(x) = xy + xyT(x) + xyD(x) - x^2y^2T(x) - x^2y^2 + \frac{xy}{1-xy}\frac{\partial P_\uparrow}{\partial x}(1) - \frac{x^2y^2}{(1-xy)^2}\left(P_\uparrow(1) - P_\uparrow(xy)\right).$$

Preuve Considérons p un élément de $P_\uparrow$. Alors, soit p est une seule cellule, soit le haut de la première colonne de p est plus haut que celui de la deuxième colonne, soit le bas de la première colonne de p est plus bas que la deuxième colonne, soit la première colonne de p ne dépasse pas la deuxième colonne ni en haut ni en bas.

La série associée au premier cas est xy. Dans le deuxième cas, p est un polyomino tas auquel on a ajouté une cellule en haut de la première colonne (à gauche). En effet, comme le parallélogramme de p est ascendant, toute colonne de p a sa cellule la plus basse à la même hauteur ou plus haute que la cellule la plus basse de toute colonne à gauche.

Dans le troisième cas, p est un dirigé convexe auquel on a ajouté une cellule en bas de la première colonne (à gauche). En effet, le fait que la cellule la plus basse de la première colonne soit plus basse que la colonne suivante permet la construction d'un parallélogramme ascendant avec possiblement un tas à droite.

Les cas 2 et 3 ne sont pas mutuellement exclusifs. Cela explique les termes

$$-x^2y^2T(x) - x^2y^2$$

qui représentent le cas où la première colonne dépasse la seconde en haut et en bas. Aussi, le terme x^2y^2 représente une colonne de 2 cellules. C'est la seule construction à l'intersection des cas 2 et 3 qui ne peut être un tas auquel on a ajouté une cellule en haut et en bas [1] (pris en compte par $x^2y^2T(x)$), car $t(0,0)$ n'est pas défini.

Pour le cas 4, voyez la figure 16. Dans ce cas, on construit d'abord tous les polyominos convexes dont le parallélogramme est ascendant avec la première colonne qui ne dépasse pas la seconde ni en haut ni en bas. Mais, en plus, on construit des polyominos qui ne sont pas convexes : le haut de la première colonne dépasse la seconde colonne (voyez le premier terme du membre de droite dans la figure 16). Ces premières constructions correspondent au terme $\frac{xy}{1-xy}\frac{\partial P_\uparrow}{\partial x}(1)$. Il faut donc, dans un deuxième temps, enlever ces polyominos dont la première colonne dépasse la deuxième (voyez le deuxième terme du membre de droite dans la figure 16). Cette deuxième étape de construction correspond au terme $-\frac{x^2y^2}{(1-xy)^2}P_\uparrow(1)$. Mais, en faisant cela, on retranche des polyominos qui n'ont pas été construits à la première étape. Ce sont des polyominos dont la première colonne dépasse en haut et en bas (voyez le troisième terme du membre de droite dans la figure 16) ; ils correspondent au terme $\frac{x^2y^2}{(1-xy)^2}P_\uparrow(xy)$.

En rassemblant les cas 1,2,3 et 4, on obtient l'équation fonctionnelle attendue.

[1] Dans le mémoire d'Ariane Robitaille [11], ce cas particulier est oublié et de ce fait, l'équation fonctionnelle associée aux polyominos convexes qu'on y trouve est fausse.

■

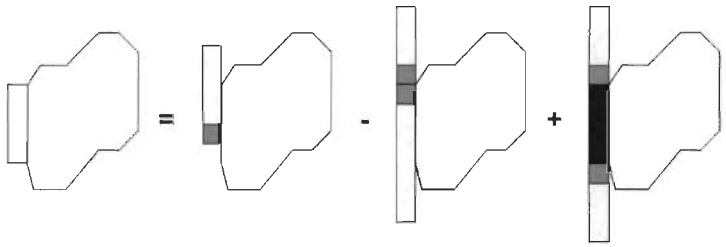

Figure 16 Construction d'un ensemble de polyominos convexes

Théorème 2.3.4 *Soit* $P_{\updownarrow}(x,y) = \sum_{m,n\geq 0} p_{\updownarrow}(m,n)x^m y^n$, *la série génératrice des polyominos convexes dont la partie qui est un polyomino parallélogramme est ascendante et descendante. Alors, l'équation fonctionnelle associée à ces polyominos est*

$$\begin{aligned} P_{\updownarrow}(x) = xy &+ 2xyT(x) - x^2y^2T(x) - x^2y^2 \\ &+ \frac{xy}{1-xy}\frac{\partial P_{\updownarrow}}{\partial x}(1) - \frac{x^2y^2}{(1-xy)^2}\left(P_{\updownarrow}(1) - P_{\updownarrow}(xy)\right). \end{aligned}$$

Preuve La preuve est la même qu'au théorème 2.3.3 sauf qu'au troisième cas, p est forcé d'être un polyomino tas auquel on a ajouté une cellule en bas de la première colonne.

■

Théorème 2.3.5 *Soit* $P_c(x,y) = \sum_{m,n\geq 0} p_c(m,n)x^m y^n$, *la série génératrice des polyominos convexes. Alors,*

$$\begin{aligned} P_c(x,y) &= 2P_{\uparrow}(x,y) - P_{\updownarrow}(x,y) \\ &= xy + 2xyD(x) - x^2y^2T(x) - x^2y^2 \\ &+ 2\left[\frac{xy}{1-xy}\frac{\partial P_{\uparrow}}{\partial x}(1) - \frac{x^2y^2}{(1-xy)^2}\left(P_{\uparrow}(1) - P_{\uparrow}(xy)\right)\right] \\ &- \left[\frac{xy}{1-xy}\frac{\partial P_{\updownarrow}}{\partial x}(1) - \frac{x^2y^2}{(1-xy)^2}\left(P_{\updownarrow}(1) - P_{\updownarrow}(xy)\right)\right]. \end{aligned}$$

Preuve Il suffit d'utiliser le principe d'inclusion exclusion, d'exploiter le fait que $|P_\uparrow| = |P_\downarrow|$ implique que $P_\uparrow(x,y) = P_\downarrow(x,y)$ et de constater que l'ensemble des polyominos convexes dont la partie qui est un polyomino parallélogramme orienté vers le haut et vers le bas est $P_\downarrow \cap P_\uparrow$.

■

CHAPITRE 3

UNE MÉTHODE PAR RÉCURRENCE

3.1 Introduction

Les objectifs de ce chapitre sont d'énumérer à l'aide d'un programme récursif les polyominos à hauteur bornée et de les construire. Ces polyominos ne sont pas nécessairement convexes. On définira un tel polyomino comme étant un ensemble 4-connexe de cellules dont les ordonnées sont plus petites qu'un entier préétabli.

Les polyominos énumérés dans ce chapitre sont dits à forme fixée. Deux polyominos à forme fixée sont égaux si et seulement si on peut obtenir un de l'autre par une translation. Un programme provenant des algorithmes, des formules et des idées de ce chapitre se trouve à l'annexe 1.

La figure 17 illustre comment est parcouru l'ensemble des constructions possibles produites par l'algorithme qui sera présenté. Sur cette figure, les cellules sont noires. La récurrence utilisée parcourt un arbre de possibilités. Sur l'illustration, on voit une partie de cet arbre dont les noeuds sont les constructions possibles. Ces constructions contribueront ou non à l'énumération des polyominos. Aussi, sur l'illustration, les constructions sont numérotées selon l'ordre dans lequel elles apparaissent dans l'application de l'algorithme qui sera détaillé dans les prochaines sections. Remarquez comment certaines cellules sont fixées (la cellule la plus basse de la deuxième colonne) en cours d'énumération et voyez comment cela contourne les nombreuses possibilités (2^6 pour les deux dernières colonnes illustrées). En effet, pour le cas illustré, la présence de la cellule la plus basse de la première colonne et l'absence de cellule ailleurs dans cette colonne force la première cellule de la deuxième colonne à apparaître dans les constructions considérées. Ce sont de telles considérations qui supportent les idées présentées dans ce chapitre.

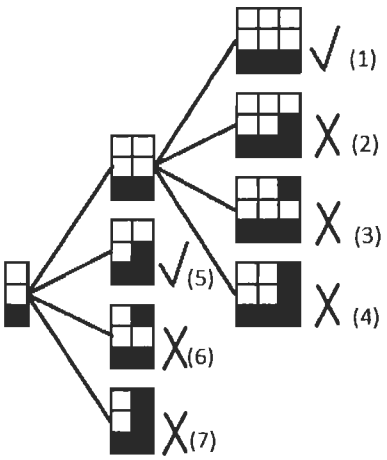

Figure 17 Construction de polyominos colonne par colonne

3.2 Définitions

On peut représenter un polyomino par un ensemble de points du premier quadrant du plan cartésien. Dans ce chapitre, les cellules les plus à gauche d'un polyomino auront 1 comme abscisse et toutes les cellules auront une ordonnée positive.

Pour alléger le texte, sauf en cas de spécification contraire, connecté et connexe signifieront 4-connecté et 4-connexe respectivement.

Définition 3.2.1 Un ***polyomino à hauteur bornée*** est un polyomino tel que l'ordonnée de chaque cellule de celui-ci est plus petite ou égale à une borne préétablie. La hauteur de ce polyomino n'est pas nécessairement égale à cette borne.

Définition 3.2.2 L'ensemble des cellules d'un polyomino ayant la même abscisse est appelé une ***colonne*** de ce polyomino.

Définition 3.2.3 Dans un polyomino, considérons E, un ensemble non vide et connexe de cellules appartenant à une même colonne C. S'il est impossible d'ajouter une cellule

de C à E tout en laissant E connexe, alors E est une composante connexe de C. On dira qu'une telle composante connexe est une ***colonne élémentaire simple***. La hauteur d'une colonne élémentaire simple est égale au nombre de cellules qui la composent. Sa position absolue est définie par l'ordonnée de sa cellule la plus basse. On notera par le couple (n, m) une colonne élémentaire simple de hauteur n à la position absolue m.

Exemple 3.2.1 La colonne élémentaire simple encadrée à la figure 18 est notée $(3, 5)$. L'ordonnée de la cellule la plus basse d'un polyomino sera toujours 1.

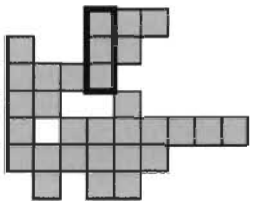

Figure 18 Colonne élémentaire simple

Définition 3.2.4 Si une cellule d'abscisse x_0 d'une colonne élémentaire simple a est connectée à une cellule d'abscisse x_0 d'une colonne élémentaire simple b par un ensemble de cellules dont les abscisses sont inférieures à x_0, alors a et b sont dites ***connectées par la gauche***. De plus, si une colonne élémentaire simple a est connectée par la gauche à une colonne élémentaire simple b et que b est connectée par la gauche à une colonne élémentaire simple c, alors a et c sont connectées par la gauche.

La connexité à gauche de colonnes élémentaires simples est une relation transitive symétrique et réflexive. C'est donc une relation d'équivalence.

Définition 3.2.5 On dira que l'ensemble des colonnes élémentaires simples d'une même colonne de polyomino connectées par la gauche forment une ***colonne élémentaire***.

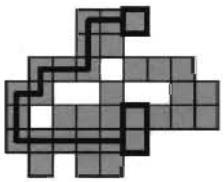

Figure 19 Une colonne élémentaire

Une colonne élémentaire est donc une classe d'équivalence pour la relation de connexité à gauche sur les colonnes élémentaires simples.

Définition 3.2.6 Les colonnes élémentaires simples qui composent une colonne élémentaire ont une ***position relative*** dans cette colonne élémentaire. Les positions relatives de ces colonnes élémentaires simples sont numérotées de bas en haut, soit 1,2,3,....

Définition 3.2.7 Dans une colonne de polyomino, les premières colonnes élémentaires simples de chaque colonne élémentaire apparaissent dans un certain ordre et on peut les ordonner selon leurs positions absolues. Les colonnes élémentaires d'une colonne de polyomino sont ordonnées de bas en haut selon l'ordre des positions absolues de leurs premières colonnes élémentaires simples. On notera 1,2,3,... les ***positions relatives*** de ces colonnes élémentaires.

Exemple 3.2.2 La cellule encadrée à la figure 20 est la colonne élémentaire simple 1 de la colonne élémentaire 3 de sa colonne de polyomino.

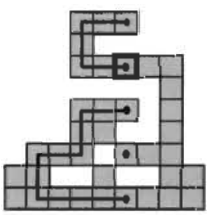

Figure 20 Position relative de colonnes élémentaires

Remarque 3.2.1 Selon les définitions vues jusqu'à présent, une colonne de polyomino peut être définie comme une suite de colonnes élémentaires qui à leur tour sont formées de suites de colonnes élémentaires simples. Comme les colonnes élémentaires simples peuvent être décrites par des couples (n, m), on peut définir une colonne de polyomino par une suite de suites de couples. △

Notation 3.2.1 Dans tout ce chapitre, on utilisera la notation

$$\vec{c}_{\omega,(\alpha,\beta)} = (n_{\omega,(\alpha,\beta)}, m_{\omega,(\alpha,\beta)})$$

pour décrire la colonne élémentaire simple β de la colonne élémentaire α de la colonne C_ω. Les valeurs $n_{\omega,(\alpha,\beta)}$ et $m_{\omega,(\alpha,\beta)}$ sont respectivement la hauteur et la position absolue de cette colonne élémentaire simple. Il est important de noter que les lettres n et m seront réservées pour décrire les hauteurs et positions absolues des colonnes élémentaires simples.

Une colonne de polyomino s'écrira $C_\omega = \{\{\vec{c}_{\omega,(i,j)}\}_{j=1}^{k_i}\}_{i=1}^{l_\omega}$ où k_i est le nombre de colonnes élémentaires simples de la colonne élémentaire i et l_ω est le nombre de colonnes élémentaires de cette colonne de polyomino. Cette même notation sera utilisée pour décrire des parties de colonnes. ▲

Exemple 3.2.3 La colonne dont les colonnes élémentaires simples sont encadrées à la figure suivante s'écrit $\{\{(1,1),(1,5)\},\{(1,3)\},\{(1,7),(2,9)\}\}$.

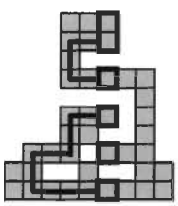

Figure 21 Une colonne

3.3 Les précolonnes

Le but de cette section est de définir et énumérer les précolonnes pour ensuite définir par extension l'ensemble des précolonnes admissibles immédiatement à droite d'une colonne donnée.

Théorème 3.3.1 *Si C_0 est une colonne d'un polyomino sans être la plus à droite de celui-ci et que C_1 est la colonne immédiatement à droite de C_0, alors pour chaque colonne élémentaire de C_0, il existe une cellule de C_1 qui a une arête commune avec une cellule de cette colonne élémentaire de C_0.*

Preuve Considérons C_0 et C_1, deux colonnes consécutives d'un polyomino P telles que C_1 est à droite de C_0 et considérons une colonne élémentaire c_0 de C_0.

Si c_0 est la seule colonne élémentaire de C_0, mais qu'aucune cellule de C_1 n'a d'arête commune avec une cellule de c_0, alors il n'existe aucun chemin de C_1 à C_0 et P n'est pas connexe, ce qui est impossible.

Si c_0 n'est pas la seule colonne élémentaire de C_0, alors considérons une colonne élémentaire c_0' de C_0 telle que $c_0' \neq c_0$. Dans ce cas, les cellules à gauche de C_1 ne suffisent pas à connecter c_0 à c_0'. Donc, il doit y avoir au moins une cellule immédiatement à droite de c_0, sinon P n'est pas connexe, car c_0 et c_0' ne sont pas connectées à gauche.

■

Selon ce dernier théorème, une colonne immédiatement à droite d'une autre doit avoir, pour chaque colonne élémentaire de cette autre colonne, au moins une cellule ayant une arête commune avec cette colonne élémentaire.

Définition 3.3.1 Soit c_1, un ensemble non vide de cellules dans la colonne immédiatement à droite d'une colonne élémentaire c_0. De plus, disons que les cellules de c_1 ont

chacune une arête commune avec une cellule de c_0. Alors, c_1 est une ***précolonne élémentaire*** de c_0 si et seulement si toute cellule à droite de c_0 ayant une arête commune avec une cellule de c_0 est une cellule de c_1.

Remarque 3.3.1 Une précolonne élémentaire a la même forme qu'une colonne élémentaire. C'est-à-dire qu'elle peut s'écrire comme $\{\vec{c}_{\omega,(a,j)}\}_{j=1}^{k_a}$. En effet, toutes ses cellules sont connectées entre elles par la gauche selon la définition 1. Elle est donc une suite de colonnes élémentaires simples. △

Définition 3.3.2 Soit une colonne C_0 d'un polyomino. Un ensemble de cellules constitué des précolonnes élémentaires de chaque colonne élémentaire de C_0 est appelé une précolonne de C_0.

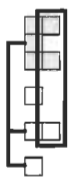

Figure 22 Une précolonne élémentaire

Remarque 3.3.2 Dans une précolonne C_1 d'une colonne C_0, une précolonne élémentaire c_1 d'une colonne élémentaire c_0 de C_0 est une colonne élémentaire de C_1. En effet, si c_1 est une précolonne élémentaire dans C_1, alors les cellules de c_1 sont connectées par la gauche entre elles selon la définition 3.3.1. De plus, si c est une cellule quelconque de $C_1 - c_1$, alors c est relié à une colonne élémentaire de C_0 différente de c_0. Les colonnes élémentaires de C_0 étant disjointe selon la définition 3.2.5, c ne peut être relié à c_0 qui est relié à c_1. Donc, c n'est pas relié à c_1 et ainsi, c_1 est maximal, car c est une cellule quelconque de $C_1 - c_1$. △

Définition 3.3.3 Soit c_0, une colonne élémentaire de C_0. Alors, le *poids* de c_0 noté $|c_0|$ est le nombre de cellules de c_0. De plus, le poids de C_0 est son nombre de cellule et on le notera $|C_0|$.

Remarque 3.3.3 Pour une colonne $C_0 = \{\{\vec{c}_{0,(i,j)}\}_{j=1}^{k_i}\}_{i=1}^{l}$, on a $|C_0| = \sum_{i=1}^{l} |c_{0,i}|$. △

Théorème 3.3.2 *En utilisant la notation 3.2.1, immédiatement à droite d'une colonne élémentaire $c_{0,a}$ s'écrivant $\{\vec{c}_{0,(a,j)}\}_{j=1}^{k_a}$, il y a $2^{|c_{0,a}|} - 1$ manières de construire une précolonne élémentaire.*

Preuve Considérons une colonne élémentaire $c_{0,a}$ s'écrivant $\{\vec{c}_{0,(a,j)}\}_{j=1}^{k_a}$ et disons que c_1 est une précolonne élémentaire qu'on veut **construire** immédiatement à droite de $c_{0,a}$. Dans ce cas, chaque cellule de c_1 doit toucher à $c_{0,a}$. Clairement, la somme des hauteurs des colonnes élémentaires simples de $c_{0,a}$ correspond au nombre de cellules de $c_{0,a}$ et cette somme est $\sum_{j=1}^{k_a} n_{0,(a,j)}$. Et, à droite de chaque cellule de $c_{0,a}$, il y a une cellule de c_1 ou non. Cela fait $2^{|c_{0,a}|}$ possibilitées, mais c_1 n'est pas vide. Il y a donc $2^{|c_{0,a}|} - 1$ manières de construire c_1.

■

Corollaire 3.3.3 *Le nombre de précolonnes d'une colonne $C_0 = \{\{\vec{c}_{0,(i,j)}\}_{j=1}^{k_i}\}_{i=1}^{l}$ est*

$$\prod_{i=1}^{l} [2^{|c_{0,i}|} - 1]$$

Preuve Il suffit d'appliquer le théorème 3.3.2 pour chacune des l colonnes élémentaires de C_0.

■

Remarque 3.3.4 Si on se réfère à la notation 3.2.1, $|c_{0,a}| = \sum_{j=1}^{k_a} n_{0,(a,j)}$. △

Remarque 3.3.5 Il existe une bijection entre les entiers positifs et les colonnes de polyomino. En effet, en parcourant une colonne de polyomino de bas en haut et en remplaçant les cellules par des 1 et les espaces par des 0, on obtient un entier positif en base 2. Il est aussi simple de transformer un entier positif en colonne de polyomino en l'écrivant en base 2. △

Il est clair que pour une colonne élémentaire $c_0 = \{\vec{c}_{0,(a,j)}\}_{j=1}^{k_a}$, certains ensembles de cellules ne peuvent constituer une précolonne élémentaire à droite de c_0. Voyons une bijection entre les précolonnes élémentaires à droite de c_0 et les entiers de 1 à $2^{|c_{0,a}|} - 1$.

Soit un nombre entier N tel que $1 \leq N \leq 2^{|c_{0,a}|} - 1$ et considérons N sous sa forme binaire. Il suffit, pour chaque chiffre (« bit ») i de N sous sa forme binaire, de placer celui-ci à la position correspondant à l'ordonnée de la i-ème cellule de c_0 (On peut numéroter les cellules de c_0 de bas en haut.). En remplaçant les 1 par des cellules, on obtient une précolonne élémentaire à droite de c_0. Inversement, à partir d'une précolonne élémentaire à droite de c_0, on écrit un entier positif en base 2 plus petit ou égale à $2^{|c_{0,a}|} - 1$ comme suit. Vis-à-vis les cellules de c_0, on écrit un 1 s'il y a une cellule de la précolonne élémentaire et, un 0 s'il n'y a pas de cellule de la précolonne élémentaire (Vis-à-vis les trous de c_0, on n'écrit rien.). La chaîne obtenue par le codage des positions à droite de c_0 donne l'entier qui correspond à la précolonne élémentaire.

On peut donc associer un l-tuple d'entiers $(N_1, N_2, ..., N_l)$ tel que $1 \leq N_i \leq 2^{|c_{0,i}|} - 1$ à une précolonne de $C_0 = \{\{\vec{c}_{0,(i,j)}\}_{j=1}^{k_i}\}_{i=1}^{l}$ dont les précolonnes élémentaires sont $c_{0,i} = \{\vec{c}_{0,(i,j)}\}_{j=1}^{k_i}$ avec $i = 1, ..., l$.

L'algorithme suivant transforme un entier positif $N < 2^{|c_{0,a}|}$ en la précolonne élémentaire correspondante sous la forme $\{\vec{c}_{1,(b,j)}\}_{j=1}^{k_b}$.

Algorithme 3.3.1 *Fabrication d'une précolonne élémentaire*

$\mathbf{FPE}\left(N, c_0 = \{\vec{c}_{0,(a,j)}\}_{j=1}^{k_a}\right)$

résultat $\leftarrow 0$;
pour i de 1 à k_a :

résultat $\leftarrow$ résultat $+ \left[\texttt{partieEntière}\left(\frac{N}{2^{\sum_{j=1}^{i-1} n_{0,(a,j)}}}\right) \mod 2^{n_{0,(a,i)}}\right] \cdot 2^{m_{0,(a,i)}-1}$

fin pour ;

colonneBinaire[] $\leftarrow$ convertirEnBinaire(résultat) ;
colonneRésultat[] ;
CES $\leftarrow 1$;
position $\leftarrow 1$;
tant que position $\leq$ taille(colonneBinaire[]) :

si colonneBinaire[position]=1, alors :

positionCES $\leftarrow$ position ;
hauteurCES $\leftarrow 0$;
tant que position $\leq$ taille(colonneBinaire[]) et colonneBinaire[position]=1 :
hauteurCES ++ ;
position ++ ;
fin tant que ;
colonneRésultat[CES] $\leftarrow$ (hauteurCES, positionCES) ;
CES ++ ;

fin si ;

position ++ ;

fin tant que ;

retourner colonneRésultat[] ;

fin.

On utilisera cet algorithme dans celui qui suit, où un l-tuple d'entiers positifs est transformé en une précolonne $C_1 = \{\{\vec{c}_{1,(i,j)}\}_{j=1}^{k_i}\}_{i=1}^{l}$. Dans l'algorithme 3.3.2, la fonction « ordonner(précolonne[]) »ordonne les précolonnes élémentaires $\mathrm{FPE}\left(\{\vec{c}_{0,(i,j)}\}_{j=1}^{k_i}\right)$

qui sont dans le tableau qu'elle reçoit. Ces précolonnes élémentaires doivent être indexées selon leurs positions absolues.

Algorithme 3.3.2 *Fabrication d'une précolonne*

$\mathbf{FP}\big((N_1, N_2, ..., N_l), C_0 = \{\{\vec{c}_{0,(i,j)}\}_{j=1}^{k_i}\}_{i=1}^{l}\big)$

précolonne[] ;
pour i de 1 à l :

précolonne[i] $\leftarrow$ FPE$\big(N_i, \{\vec{c}_{0,(i,j)}\}_{j=1}^{k_i}\big)$;

fin pour ;
ordonner(précolonne[]) ;
retourner précolonne[] ;
fin.

Ainsi, pour décrire en extension l'ensemble des précolonnes de

$$C_0 = \{\{\vec{c}_{0,(i,j)}\}_{j=1}^{k_i}\}_{i=1}^{l},$$

il suffit de parcourir l'ensemble des l-tuplets d'entiers $(N_1, N_2, ..., N_l)$ tel que $1 \leq N_i \leq 2^{|c_{0,i}|} - 1$ et d'utiliser l'algorithme 3.3.2.

3.4 Opérations sur des colonnes

Cette section portera sur l'énumération et la construction des colonnes pouvant être placées immédiatement à droite d'une colonne donnée dans un polyomino. On définira des opérateurs permettant de calculer la colonne obtenue par « l'imbrication »d'une colonne dans une autre.

Définition 3.4.1 Soit une colonne C_0 et deux colonnes élémentaires simples a et b de C_0. Alors, a et b sont ***voisines*** si et seulement si aucune colonne élémentaire simple de C_0 n'a une position absolue entre les positions absolues de a et b.

Définition 3.4.2 Une ***colonne admissible immédiatement à droite*** d'une colonne C_0 est telle que les colonnes élémentaires de C_0 pourront être connectées entre elles par un ensemble de cellules dans les colonnes suivantes à droite de C_0.

Remarque 3.4.1 On peut caractériser les colonnes admissibles immédiatement à droite d'une colonne C_0. En effet, par la définition 2, une colonne C_1 peut être immédiatement à droite de C_0 si et seulement si chaque colonne élémentaire de C_0 a une arête commune avec une cellule de C_1. △

Définition 3.4.3 Considérons C_0, une colonne.

- L'intervalle $[y_a+1, y_b-1]$ est un ***vide élémentaire*** entre les colonnes élémentaires simples a et b si et seulement si a et b sont deux colonnes élémentaires simples voisines dans C_0 telles que la position absolue de a est inférieure à celle de b, y_a est l'ordonnée de la plus haute cellule de a et y_b est l'ordonnée de la plus basse cellule de b.
- L'intervalle $[1, y_0 - 1]$ est un ***vide élémentaire*** sous c_0 si et seulement si c_0 est la colonne élémentaire simple la plus basse de C_0, y_0 est l'ordonnée de la cellule la plus basse de c_0 et $y_0 > 1$.
- L'intervalle $[y_1 + 1, B]$ est un ***vide élémentaire*** au dessus de c_1 si et seulement si c_1 est la colonne élémentaire simple dont la position absolue est la plus grande dans C_0, y_1 est l'ordonnée de la cellule la plus haute de c_1 et $y_1 < B$ où B est la borne préétablie d'un polyomino à hauteur bornée (Voir la définition 1.) duquel C_0 est une colonne.

Définition 3.4.4 Il est possible d'ordonner de bas en haut les vides élémentaires d'une colonne. On les notera I_1, I_2,... selon l'ordre dans lequel ils se présentent dans leur colonne lorsqu'on parcourt celle-ci de bas en haut.

Définition 3.4.5 Soit une colonne C_0 et I, un vide élémentaire de C_0. De plus, considérons un ensemble C_1 de cellules dans la colonne immédiatement à droite de C_0. Alors, C_1 est une ***colonne complémentaire*** associée à I si et seulement si les ordonnées de toutes les cellules de C_1 sont dans l'intervalle I et qu'il n'y pas, immédiatement à droite de C_0, d'autres cellules que celles de C_1 ayant une ordonnée dans I.

Remarque 3.4.2 Une colonne complémentaire peut être vide. De plus, si $C_0 = \{\{\vec{c}_{0,(i,j)}\}_{j=1}^{k_i}\}_{i=1}^{l}$ est une colonne quelconque d'un polyomino, alors toute colonne immédiatement à droite de C_0 est l'union disjointe de colonnes complémentaires et d'une précolonne. De plus, en supposant que la hauteur du polyomino est bornée par un entier B, on trouve qu'il y a $2^{B-|C_0|}$ façons de construire les colonnes complémentaires de C_0. Si les colonnes élémentaires de C_0 sont notées $c_{0,i}$ $(i = 1, ..., l)$, alors par le corollaire 3.3.3 et le principe de multiplication, il y a $2^{B-|C_0|} \cdot \prod_{i=1}^{l} \left[2^{|c_{0,i}|} - 1\right]$ manières de construire une colonne immédiatement à droite de C_0. △

Dans le but de définir par extension l'ensemble $\{C_x\}$ des colonnes pouvant être placées immédiatement à droite d'une colonne C_0, on construira chaque C_x avec une précolonne C_{C_0} de C_0 et un ensemble $\{C_{I_1}, C_{I_2}, ..., C_{I_N}\}$ de colonnes complémentaires de C_0 associées aux N vides élémentaires de C_0. Les cellules de $C_{C_0} \cup \bigcup_{i=1}^{N} C_{I_i}$ forment alors une colonne pouvant suivre C_0. La précolonne C_{C_0} assure que les colonnes élémentaires de C_0 pourront être éventuellement connectées au reste du polyomino en construction.

Théorème 3.4.1 *Si $\{C_X\}$ est l'ensemble des précolonnes de C_0 et*

$\{\{C_{I_1}, C_{I_2}, ..., C_{I_N}\}_Y\}$ est l'ensemble des ensembles de colonnes complémentaires de C_0 associées aux N vides élémentaires de C_0, alors l'ensemble

$$\{C_X\} \times \{\{C_{I_1}, C_{I_2}, ..., C_{I_N}\}_Y\}$$

est en bijection avec l'ensemble des colonnes admissibles immédiatement à droite de C_0.

Preuve Soit

$$(C_{X1}, \{C_{I_1}, C_{I_2}, ..., C_{I_N}\}_{Y1}),$$
$$(C_{X2}, \{C_{I_1}, C_{I_2}, ..., C_{I_N}\}_{Y2}) \in \{C_X\} \times \{\{C_{I_1}, C_{I_2}, ..., C_{I_N}\}_Y\} \text{ et}$$
$$(C_{X1}, \{C_{I_1}, C_{I_2}, ..., C_{I_N}\}_{Y1}) \neq (C_{X2}, \{C_{I_1}, C_{I_2}, ..., C_{I_N}\}_{Y2}).$$

Alors $C_{X1} \neq C_{X2}$ ou $\{C_{I_1}, C_{I_2}, ..., C_{I_N}\}_{Y1} \neq \{C_{I_1}, C_{I_2}, ..., C_{I_N}\}_{Y2}$.

Si $C_{X1} \neq C_{X2}$, alors pour une ordonnée fixée, il y a une cellule c de C_{X1} qui a cette ordonnée, mais aucune cellule de C_{X2} n'a cette ordonnée. De plus, c a une arête commune avec une cellule de C_0 par la définition de précolonne. Alors, aucune cellule de $\{C_{I_1}, C_{I_2}, ..., C_{I_N}\}_{Y2}$ ne peut avoir cette ordonnée par la définition de colonne complémentaire. Ainsi, aucune cellule de $C_{X2} \cup \{C_{I_1}, C_{I_2}, ..., C_{I_N}\}_{Y2}$ n'a l'ordonnée de c. Donc, les colonnes formées par $(C_{X1}, \{C_{I_1}, C_{I_2}, ..., C_{I_N}\}_{Y1})$ et
$(C_{X2}, \{C_{I_1}, C_{I_2}, ..., C_{I_N}\}_{Y2})$ sont distinctes quand $C_{X1} \neq C_{X2}$.

On montre de manière similaire que

$$(C_{X1}, \{C_{I_1}, C_{I_2}, ..., C_{I_N}\}_{Y1})$$

et $(C_{X2}, \{C_{I_1}, C_{I_2}, ..., C_{I_N}\}_{Y2})$ sont distinctes quand

$$\{C_{I_1}, C_{I_2}, ..., C_{I_N}\}_{Y1} \neq \{C_{I_1}, C_{I_2}, ..., C_{I_N}\}_{Y2}.$$

Ainsi, la construction d'une colonne immédiatement à droite de C_0 en utilisant un élément de $\{C_X\} \times \{\{C_{I_1}, C_{I_2}, ..., C_{I_N}\}_Y\}$ est injective.

Pour vérifier que chaque élément de $\{C_X\} \times \{\{C_{I_1}, C_{I_2}, ..., C_{I_N}\}_Y\}$ correspond à une et une seule colonne C_x pouvant être placée immédiatement à droite de C_0, il reste à s'assurer que toute colonne C_x peut être construite à partir d'un élément de $\{C_X\} \times \{\{C_{I_1}, C_{I_2}, ..., C_{I_N}\}_Y\}$.

Considérons une colonne C_1 pouvant être immédiatement à droite de C_0. Alors, pour chaque colonne élémentaire de C_0, au moins une cellule de C_1 a une arête commune avec une cellule de cette colonne élémentaire par le théorème 3.3.1. Pour chaque colonne élémentaire de C_0, on peut regrouper les cellules de C_1 qui ont une arête commune avec cette colonne élémentaire. L'ensemble (non vide par le théorème 3.3.1) ainsi formé est une précolonne élémentaire de cette colonne élémentaire de C_0. Il suffit de rassembler les précolonnes élémentaires ainsi construites pour toutes les colonnes élémentaires de C_0 pour obtenir une précolonne C_X de C_0. Alors, les cellules de C_X forment un sous-ensemble de C_1 et les cellules de $C_1 - C_X$ ont toutes une ordonnée dans un vide complémentaire de C_0. On peut donc, pour chaque i, former une colonne complémentaire associée à I_i avec les cellules de $C_1 - C_X$ dont l'ordonnée est dans I_i. Les colonnes complémentaires obtenues constituent l'ensemble $\{C_{I_1}, C_{I_2}, ..., C_{I_N}\}_Y$ tel que $\{C_X\} \times \{\{C_{I_1}, C_{I_2}, ..., C_{I_N}\}_Y\}$ correspond à C_1.

■

Il est donc possible de construire chaque colonne pouvant suivre une colonne donnée en parcourant l'ensemble $\{C_X\} \times \{\{C_{I_1}, C_{I_2}, ..., C_{I_N}\}_Y\}$. Mais, il sera nécessaire d'écrire les colonnes ainsi obtenues comme $\{\{\vec{c}_{0,(i,j)}\}_{j=1}^{k_i}\}_{i=1}^{l}$ pour appliquer, à la prochaine section, la récurrence permettant d'énumérer les polyominos à hauteur bornée. Le reste de la présente section a pour but d'exprimer un élément de $\{C_X\} \times \{\{C_{I_1}, C_{I_2}, ..., C_{I_N}\}_Y\}$ sous la forme $\{\{\vec{c}_{0,(i,j)}\}_{j=1}^{k_i}\}_{i=1}^{l}$. L'algorithme 3.3.2 permet d'écrire un élément de $\{C_X\}$

comme $C_X = \{\{\vec{c}_{X,(i,j)}\}_{j=1}^{k_i}\}_{i=1}^{l}$. Les colonnes complémentaires peuvent également être écrites de cette même manière, elles prennent la forme $C_Y = \{\{\vec{c}_{Y,(\iota,1)}\}\}_{\iota=1}^{h}$. Il reste à calculer la colonne obtenue si on insère une colonne C_Y dans un vide élémentaire de C_X pouvant contenir C_Y.

Définition 3.4.6 Soit $C_X = \{\{\vec{c}_{X,(i,j)}\}_{j=1}^{k_i}\}_{i=1}^{l}$. Considérons aussi $\vec{c}_{X,\overline{(a,b)}}$ et $\vec{c}_{X,\underline{(a,b)}}$, des colonnes élémentaires simples voisines dans C_X telles que $m_{X,\overline{(a,b)}} \leq m_{X,\underline{(a,b)}}$. De plus, considérons $C_Y = \{\{\vec{c}_{Y,(\iota,1)}\}\}_{\iota=1}^{h}$ où toute cellule de C_Y a une ordonnée y telle que $n_{X,\overline{(a,b)}} + m_{X,\overline{(a,b)}} \leq y < m_{X,\underline{(a,b)}}$. Alors, C_Y peut s'***imbriquer*** dans C_X entre $\vec{c}_{X,\overline{(a,b)}}$ et $\vec{c}_{X,\underline{(a,b)}}$.

Remarque 3.4.3 Selon les définitions 5 et 6, toute colonne complémentaire C_{I_a} d'une colonne C_0 peut s'imbriquer dans une précolonne C_X de C_0 pour former une colonne admissible immédiatement à droite de C_0. △

Définition 3.4.7 Soit des colonnes élémentaires distinctes c_1 et c_2 d'une même colonne. Alors, c_1 ***est à l'intérieur de*** c_2 si et seulement si les ordonnées des cellules de c_1 sont toutes entre les positions absolues de deux colonnes élémentaires simples consécutives de c_2.

Définition 3.4.8 On distinguera 4 types de colonnes élémentaires simples voisines. Considérons le couple $(\vec{c}_1, \vec{c}_2)$, voisines et telles que la position absolue de $\vec{c}_1$ est plus petite que celle de $\vec{c}_2$.

- Si $\vec{c}_1$ et $\vec{c}_2$ sont dans une même colonne élémentaire, alors on dira qu'elles sont ***voisines de type 1***.
- Si $\vec{c}_1$ et $\vec{c}_2$ ne sont pas de type 1, disons que $\vec{c}_1$ et $\vec{c}_2$ sont dans les colonnes élémentaires c_1 et c_2 respectivement. Alors, $\vec{c}_1$ et $\vec{c}_2$ sont ***voisines de type 2*** si et seulement si c_1 et c_2 ne sont pas telles que l'une est à l'intérieur de l'autre.

- Si $\vec{c}_1$ et $\vec{c}_2$ ne sont ni de type 1 ni de type 2, alors elles sont dans des colonnes élémentaires distinctes, disons c_1 et c_2 respectivement, qui sont l'une à l'intérieur de l'autre. Dans ce cas, $\vec{c}_1$ et $\vec{c}_2$ sont ***voisines de type 3*** (resp. ***type 4***) si et seulement si c_2 est à l'intérieur de c_1 (resp. c_1 est à l'intérieur de c_2).

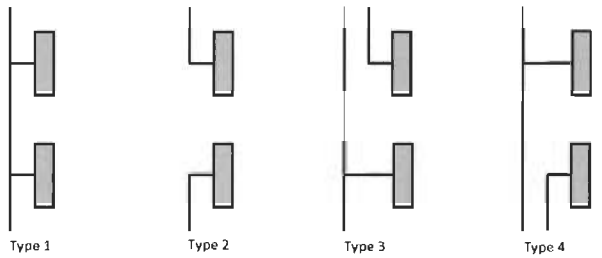

Figure 23 Des colonnes élémentaires simples voisines de chaque type

Les définitions 3.4.9 à 3.4.17 sont toutes sous les hypothèses et les notations suivantes : $C_X = \{\{\vec{c}_{X,(i,j)}\}_{j=1}^{k_i}\}_{i=1}^{l}$ avec $\vec{c}_{X,(a,b)}$ et $\vec{c}_{X,(\alpha,\beta)}$ voisines. De plus, $C_Y = \{\{\vec{c}_{Y,(\iota,1)}\}\}_{\iota=1}^{h}$ peut s'imbriquer dans C_X entre $\vec{c}_{X,(a,b)}$ et $\vec{c}_{X,(\alpha,\beta)}$ telles que $m_{X,(a,b)} < m_{X(\alpha,\beta)}$. Dans le calcul des colonnes pouvant suivre une colonne C_0, C_X sera formée d'une précolonne de C_0 et possiblement de colonnes complémentaires de C_0. Et, C_Y sera une colonne complémentaire de C_0. Pour visualiser les opérations qui seront définies, on peut se référer à la figure 23.

Définition 3.4.9 Supposons qu'aucune cellule de C_X n'a d'arête commune avec toute cellule de C_Y. Alors,

$$C_X +_{\updownarrow} C_Y = \begin{cases} \{\{\vec{c}_{X,(1,j)}\}_{j=1}^{k_1}, \{\vec{c}_{X,(2,j)}\}_{j=1}^{k_2}, \ldots, \{\vec{c}_{X,(\gamma,j)}\}_{j=1}^{k_\gamma}, \\ \{\vec{c}_{Y,(1,1)}\}, \{\vec{c}_{Y,(2,1)}\}, \ldots, \{\vec{c}_{Y,(h,1)}\}, \\ \{\vec{c}_{X,(\gamma+1,j)}\}_{j=1}^{k_{\gamma+1}}, \{\vec{c}_{X,(\gamma+2,j)}\}_{j=1}^{k_{\gamma+2}}, \ldots, \{\vec{c}_{X,(l,j)}\}_{j=1}^{k_l}\} \end{cases}$$

où $\{\vec{c}_{X,(\gamma,j)}\}_{j=1}^{k_\gamma}$ est la colonne élémentaire de C_X dont l'ordonnée de la cellule la plus basse est la plus près de $m_{Y,(1,1)}$ sans l'excéder.

Définition 3.4.10 Supposons que la cellule la plus basse de C_Y a une arête commune avec la cellule la plus haute de $\vec{c}_{X,(a,b)}$ sans que la cellule la plus haute de C_Y n'ait une arête commune avec la cellule la plus basse de $\vec{c}_{X,(\alpha,\beta)}$. Alors,

$$C_X +_{\underline{\updownarrow}} C_Y = C'_X +_{\updownarrow} \{\{\vec{c}_{Y,(2,1)}\}, \{\vec{c}_{Y,(3,1)}\}, ..., \{\vec{c}_{Y,(h,1)}\}\}$$

où

$$C'_X = \begin{cases} \{\{\vec{c}_{X,(1,j)}\}_{j=1}^{k_1}, \{\vec{c}_{X,(2,j)}\}_{j=1}^{k_2}, ..., \{\vec{c}_{X,(a,1)}, \\ \vec{c}_{X,(a,2)}, ..., \vec{c}_{X,(a,b-1)}, \vec{c}_{X'}, \vec{c}_{X,(a,b+1)}, ..., \vec{c}_{X,(a,k_a)}\}, \\ \{\vec{c}_{X,(a+1,j)}\}_{j=1}^{k_{a+1}}, \{\vec{c}_{X,(a+2,j)}\}_{j=1}^{k_{a+2}}, ..., \{\vec{c}_{X,(l,j)}\}_{j=1}^{k_l}\} \end{cases}$$

avec $\vec{c}_{X'} = (n_{X,(a,b)} + n_{Y,(1,1)}, m_{X,(a,b)})$.

Définition 3.4.11 Supposons que la cellule la plus haute de C_Y a une arête commune avec la cellule la plus basse de $\vec{c}_{X,(\alpha,\beta)}$ sans que la cellule la plus basse de C_Y n'ait une arête commune avec la cellule la plus haute de $\vec{c}_{X,(a,b)}$. Alors,

$$C_X +_{\overline{\updownarrow}} C_Y = C'_X +_{\updownarrow} \{\{\vec{c}_{Y,(1,1)}\}, \{\vec{c}_{Y,(2,1)}\}, ..., \{\vec{c}_{Y,(h-1,1)}\}\}$$

où

$$C'_X = \begin{cases} \{\{\vec{c}_{X,(1,j)}\}_{j=1}^{k_1}, \{\vec{c}_{X,(2,j)}\}_{j=1}^{k_2}, ..., \{\vec{c}_{X,(\alpha,1)}, \\ \vec{c}_{X,(\alpha,2)}, ..., \vec{c}_{X,(\alpha,\beta-1)}, \vec{c}_{X'}, \vec{c}_{X,(\alpha,\beta+1)}, ..., \vec{c}_{X,(\alpha,k_\alpha)}\}, \\ \{\vec{c}_{X,(\alpha+1,j)}\}_{j=1}^{k_{\alpha+1}}, \{\vec{c}_{X,(\alpha+2,j)}\}_{j=1}^{k_{\alpha+2}}, ..., \{\vec{c}_{X,(l,j)}\}_{j=1}^{k_l}\} \end{cases}$$

avec $\vec{c}_{X}\prime = (n_{X,(\alpha,\beta)} + n_{Y,(h,1)}, m_{Y,(h,1)})$.

Définition 3.4.12 Supposons que la cellule la plus haute de C_Y a une arête commune avec la cellule la plus basse de $\vec{c}_{X,(\alpha,\beta)}$, la cellule la plus basse de C_Y a une arête commune avec la cellule la plus haute de $\vec{c}_{X,(a,b)}$ et $h > 1$. Alors,

$$C_X +_{\underline{\underline{1}}} C_Y = C'_X +_{\underline{1}} \{\{\vec{c}_{Y,(1,1)}\}, \{\vec{c}_{Y,(2,1)}\}, ..., \{\vec{c}_{Y,(h-1,1)}\}\}$$

où

$$C'_X = \begin{cases} \{\{\vec{c}_{X,(1,j)}\}_{j=1}^{k_1}, \{\vec{c}_{X,(2,j)}\}_{j=1}^{k_2}, ..., \{\vec{c}_{X,(\alpha,1)}, \\ \vec{c}_{X,(\alpha,2)}, ..., \vec{c}_{X,(\alpha,\beta-1)}, \vec{c}_{X}\prime, \vec{c}_{X,(\alpha,\beta+1)}, ..., \vec{c}_{X,(\alpha,k_\alpha)}\}, \\ \{\vec{c}_{X,(\alpha+1,j)}\}_{j=1}^{k_{\alpha+1}}, \{\vec{c}_{X,(\alpha+2,j)}\}_{j=1}^{k_{\alpha+2}}, ..., \{\vec{c}_{X,(l,j)}\}_{j=1}^{k_l}\} \end{cases}$$

avec $\vec{c}_{X}\prime = (n_{X,(\alpha,\beta)} + n_{Y,(h,1)}, m_{Y,(h,1)})$.

Pour les définitions 13 à 17, supposons que la cellule la plus haute de C_Y a une arête commune avec la cellule la plus basse de $\vec{c}_{X,(\alpha,\beta)}$ et que la cellule la plus basse de C_Y a une arête commune avec la cellule la plus haute de $\vec{c}_{X,(a,b)}$ avec $h = 1$.

Définition 3.4.13 Supposons que $\vec{c}_{X,(a,b)}$ et $\vec{c}_{X,(\alpha,\beta)}$ sont voisines de type 1. Alors, $\vec{c}_{X,(\alpha,\beta)} = \vec{c}_{X,(a,b+1)}$ et

$$C_X +_{\circ 1} C_Y = \begin{cases} \{\{\vec{c}_{X,(1,j)}\}_{j=1}^{k_1}, \{\vec{c}_{X,(2,j)}\}_{j=1}^{k_2}, ..., \{\vec{c}_{X,(a,1)}, \\ \vec{c}_{X,(a,2)}, ..., \vec{c}_{X,(a,b-1)}, \vec{c}_{X}\prime, \vec{c}_{X,(a,b+2)}, ..., \vec{c}_{X,(a,k_a)}\}, \\ \{\vec{c}_{X,(a+1,j)}\}_{j=1}^{k_{a+1}}, \{\vec{c}_{X,(a+2,j)}\}_{j=1}^{k_{a+2}}, ..., \{\vec{c}_{X,(l,j)}\}_{j=1}^{k_l}\} \end{cases}$$

où $\vec{c}_{X}\prime = (n_{X,(a,b)} + n_{Y,(1,1)} + n_{X,(a,b+1)}, m_{X,(a,b)})$.

Définition 3.4.14 Supposons que $\vec{c}_{X,(a,b)}$ et $\vec{c}_{X,(\alpha,\beta)}$ sont voisines de type 2. Alors, $\vec{c}_{X,(a,b)} = \vec{c}_{X,(a,k_a)}$, $\vec{c}_{X,(\alpha,\beta)} = \vec{c}_{X,(\alpha,1)}$ et

$$C_X +_{o2} C_Y = \begin{cases} \{\{\vec{c}_{X,(1,j)}\}_{j=1}^{k_1}, \{\vec{c}_{X,(2,j)}\}_{j=1}^{k_2}, ..., \{\vec{c}_{X,(a,1)}, \\ \vec{c}_{X,(a,2)}, ..., \vec{c}_{X,(a,k_a-1)}, \vec{c}_{X'}, \vec{c}_{X,(\alpha,2)}, \vec{c}_{X,(\alpha,3)}, ..., \vec{c}_{X,(\alpha,k_\alpha)}\}, \\ \{\vec{c}_{X,(a+2,j)}\}_{j=1}^{k_{a+2}}, \{\vec{c}_{X,(a+3,j)}\}_{j=1}^{k_{a+3}}, ..., \{\vec{c}_{X,(l,j)}\}_{j=1}^{k_l}\} \\ \\ \text{si } \alpha = a+1, \\ \\ \{\{\vec{c}_{X,(1,j)}\}_{j=1}^{k_1}, \{\vec{c}_{X,(2,j)}\}_{j=1}^{k_2}, ..., \{\vec{c}_{X,(a,1)}, \\ \vec{c}_{X,(a,2)}, ..., \vec{c}_{X,(a,k_a-1)}, \vec{c}_{X'}, \vec{c}_{X,(\alpha,2)}, \vec{c}_{X,(\alpha,3)}, ..., \vec{c}_{X,(\alpha,k_\alpha)}\}, \\ \{\vec{c}_{X,(a+1,j)}\}_{j=1}^{k_{a+1}}, \{\vec{c}_{X,(a+2,j)}\}_{j=1}^{k_{a+2}}, ..., \{\vec{c}_{X,(\alpha-1,j)}\}_{j=1}^{k_{\alpha-1}}, \\ \{\vec{c}_{X,(\alpha+1,j)}\}_{j=1}^{k_{\alpha+1}}, \{\vec{c}_{X,(\alpha+2,j)}\}_{j=1}^{k_{\alpha+2}}, ..., \{\vec{c}_{X,(l,j)}\}_{j=1}^{k_l}\} \\ \\ \text{sinon} \end{cases}$$

où $\vec{c}_{X'} = (n_{X,(a,b)} + n_{Y,(1,1)} + n_{X,(\alpha,\beta)}, m_{X,(a,b)})$.

Définition 3.4.15 Supposons que $\vec{c}_{X,(a,b)}$ et $\vec{c}_{X,(\alpha,\beta)}$ sont voisines de type 3. Alors, $\vec{c}_{X,(\alpha,\beta)} = \vec{c}_{X,(\alpha,1)}$ et les $\vec{c}_{X,(\alpha,j)}$, $(j = 1, ..., k_\alpha)$ ont toutes une position absolue entre $m_{X,(a,b)}$ et $m_{X,(a,b+1)}$. De plus,

$$C_X +_{o3} C_Y = \begin{cases} \{\{\vec{c}_{X,(1,j)}\}_{j=1}^{k_1}, \{\vec{c}_{X,(2,j)}\}_{j=1}^{k_2}, ..., \{\vec{c}_{X,(a,1)}, \\ \vec{c}_{X,(a,2)}, ..., \vec{c}_{X,(a,b-1)}, \vec{c}_{X}\prime, \vec{c}_{X,(\alpha,2)}, \vec{c}_{X,(\alpha,3)}, ..., \vec{c}_{X,(\alpha,k_\alpha)}, \\ \vec{c}_{X,(a,b+1)}, \vec{c}_{X,(a,b+2)}, ..., \vec{c}_{X,(a,k_a)}\}, \\ \{\vec{c}_{X,(a+2,j)}\}_{j=1}^{k_{a+2}}, \{\vec{c}_{X,(a+3,j)}\}_{j=1}^{k_{a+3}}, ..., \{\vec{c}_{X,(l,j)}\}_{j=1}^{k_l}\} \\ \\ \text{si } \alpha = a+1, \\ \\ \{\{\vec{c}_{X,(1,j)}\}_{j=1}^{k_1}, \{\vec{c}_{X,(2,j)}\}_{j=1}^{k_2}, ..., \{\vec{c}_{X,(a,1)}, \\ \vec{c}_{X,(a,2)}, ..., \vec{c}_{X,(a,b-1)}, \vec{c}_{X}\prime, \vec{c}_{X,(\alpha,2)}, \vec{c}_{X,(\alpha,3)}, ..., \vec{c}_{X,(\alpha,k_\alpha)}, \\ \vec{c}_{X,(a,b+1)}, \vec{c}_{X,(a,b+2)}, ..., \vec{c}_{X,(a,k_a)}\}, \\ \{\vec{c}_{X,(a+1,j)}\}_{j=1}^{k_{a+1}}, \{\vec{c}_{X,(a+2,j)}\}_{j=1}^{k_{a+2}}, ..., \{\vec{c}_{X,(\alpha-1,j)}\}_{j=1}^{k_{\alpha-1}}, \\ \{\vec{c}_{X,(\alpha+1,j)}\}_{j=1}^{k_{\alpha+1}}, \{\vec{c}_{X,(\alpha+2,j)}\}_{j=1}^{k_{\alpha+2}}, ..., \{\vec{c}_{X,(l,j)}\}_{j=1}^{k_l}\} \\ \\ \text{sinon} \end{cases}$$

où $\vec{c}_X\prime = (n_{X,(a,b)} + n_{Y,(1,1)} + n_{X,(\alpha,\beta)}, m_{X,(a,b)})$.

Définition 3.4.16 Supposons que $\vec{c}_{X,(a,b)}$ et $\vec{c}_{X,(\alpha,\beta)}$ sont voisines de type 4. Alors, $\vec{c}_{X,(a,b)} = \vec{c}_{X,(a,k_a)}$ et les $\vec{c}_{X,(a,j)}$, $(j = 1, ..., k_a)$ ont toutes une position absolue entre $m_{X,(\alpha,\beta-1)}$ et $m_{X,(\alpha,\beta)}$. De plus,

$$C_X +_{o4} C_Y = \begin{cases} \{\{\vec{c}_{X,(1,j)}\}_{j=1}^{k_1}, \{\vec{c}_{X,(2,j)}\}_{j=1}^{k_2}, ..., \{\vec{c}_{X,(\alpha,1)}, \\ \vec{c}_{X,(\alpha,2)}, ..., \vec{c}_{X,(\alpha,\beta-1)}, \vec{c}_{X,(a,1)}, \vec{c}_{X,(a,2)}, ..., \vec{c}_{X,(a,b-1)}, \vec{c}_X\prime, \\ \vec{c}_{X,(\alpha,\beta+1)}, \vec{c}_{X,(\alpha,\beta+2)}, ..., \vec{c}_{X,(\alpha,k_\alpha)}\}, \\ \{\vec{c}_{X,(\alpha+2,j)}\}_{j=1}^{k_{\alpha+2}}, \{\vec{c}_{X,(\alpha+3,j)}\}_{j=1}^{k_{\alpha+3}}, ..., \{\vec{c}_{X,(l,j)}\}_{j=1}^{k_l}\} \\ \\ \text{si } a = \alpha + 1, \\ \\ \{\{\vec{c}_{X,(1,j)}\}_{j=1}^{k_1}, \{\vec{c}_{X,(2,j)}\}_{j=1}^{k_2}, ..., \{\vec{c}_{X,(\alpha,1)}, \\ \vec{c}_{X,(\alpha,2)}, ..., \vec{c}_{X,(\alpha,\beta-1)}, \vec{c}_{X,(a,1)}, \vec{c}_{X,(a,2)}, ..., \vec{c}_{X,(a,b-1)}, \vec{c}_X\prime, \\ \vec{c}_{X,(\alpha,\beta+1)}, \vec{c}_{X,(\alpha,\beta+2)}, ..., \vec{c}_{X,(\alpha,k_\alpha)}\}, \\ \{\vec{c}_{X,(\alpha+1,j)}\}_{j=1}^{k_{\alpha+1}}, \{\vec{c}_{X,(\alpha+2,j)}\}_{j=1}^{k_{\alpha+2}}, ..., \{\vec{c}_{X,(a-1,j)}\}_{j=1}^{k_{a-1}}, \\ \{\vec{c}_{X,(a+1,j)}\}_{j=1}^{k_{a+1}}, \{\vec{c}_{X,(a+2,j)}\}_{j=1}^{k_{a+2}}, ..., \{\vec{c}_{X,(l,j)}\}_{j=1}^{k_l}\} \\ \\ \text{sinon} \end{cases}$$

où $\vec{c}_X\prime = (n_{X,(a,b)} + n_{Y,(1,1)} + n_{X,(\alpha,\beta)}, m_{X,(a,b)})$.

Définition 3.4.17

$$C_X +_o C_Y = \begin{cases} C_X +_{o1} C_Y \text{ si } \vec{c}_{X,(a,b)} \text{ et } \vec{c}_{X,(\alpha,\beta)} \text{ sont voisines de type 1,} \\ C_X +_{o2} C_Y \text{ si } \vec{c}_{X,(a,b)} \text{ et } \vec{c}_{X,(\alpha,\beta)} \text{ sont voisines de type 2,} \\ C_X +_{o3} C_Y \text{ si } \vec{c}_{X,(a,b)} \text{ et } \vec{c}_{X,(\alpha,\beta)} \text{ sont voisines de type 3,} \\ C_X +_{o4} C_Y \text{ si } \vec{c}_{X,(a,b)} \text{ et } \vec{c}_{X,(\alpha,\beta)} \text{ sont voisines de type 4.} \end{cases}$$

Définition 3.4.18 Considérons $C_X = \{\{\vec{c}_{X,(i,j)}\}_{j=1}^{k_i}\}_{i=1}^{l}$ et $C_Y = \{\{\vec{c}_{Y,(\iota,1)}\}\}_{\iota=1}^{h}$ telles que toutes les cellules de C_Y ont une ordonnée inférieure aux ordonnées des cellules de C_X. De plus, supposons qu'aucune cellule de C_Y n'a d'arête commune avec toute

cellule de C_X. Alors,

$$C_X +_{\downarrow} C_Y = \begin{cases} \{\{\vec{c}_{Y,(1,1)}\}, \{\vec{c}_{Y,(2,1)}\}, ..., \{\vec{c}_{Y,(h,1)}\}, \\ \{\vec{c}_{X,(1,j)}\}_{j=1}^{k_1}, \{\vec{c}_{X,(2,j)}\}_{j=1}^{k_2}, ..., \{\vec{c}_{X,(l,j)}\}_{j=1}^{k_l}\}. \end{cases}$$

Définition 3.4.19 Considérons $C_X = \{\{\vec{c}_{X,(i,j)}\}_{j=1}^{k_i}\}_{i=1}^{l}$ et $C_Y = \{\{\vec{c}_{Y,(\iota,1)}\}\}_{\iota=1}^{h}$ telles que toutes les cellules de C_Y ont une ordonnée inférieure aux ordonnées des cellules de C_X. De plus, supposons que la cellule la plus haute de C_Y a une arête commune avec la cellule la plus basse de C_X. Alors,

$$C_X +_{\overline{\downarrow}} C_Y = C_X\prime +_{\downarrow} \{\{\vec{c}_{Y,(1,1)}\}, \{\vec{c}_{Y,(2,1)}\}, ..., \{\vec{c}_{Y,(h-1,1)}\}\}$$

où

$$C_X\prime = \{\{\vec{c}_X\prime, \vec{c}_{X,(1,2)}, \vec{c}_{X,(1,3)}, ..., \vec{c}_{X,(1,k_1)}\}, \{\vec{c}_{X,(2,j)}\}_{j=1}^{k_2}, \{\vec{c}_{X,(3,j)}\}_{j=1}^{k_3}, ..., \{\vec{c}_{X,(l,j)}\}_{j=1}^{k_l}\}$$

avec $\vec{c}_X\prime = (n_{Y,(h,1)} + n_{X,(1,1)}, m_{Y,(h,1)})$.

Définition 3.4.20 Considérons $C_X = \{\{\vec{c}_{X,(i,j)}\}_{j=1}^{k_i}\}_{i=1}^{l}$ et $C_Y = \{\{\vec{c}_{Y,(\iota,1)}\}\}_{\iota=1}^{h}$ telles que toutes les cellules de C_Y ont une ordonnée plus grande que les ordonnées des cellules de C_X. De plus, supposons qu'aucune cellule de C_Y n'a d'arête commune avec toute cellule de C_X. Alors,

$$C_X +_{\uparrow} C_Y = \begin{cases} \{\{\vec{c}_{X,(1,j)}\}_{j=1}^{k_1}, \{\vec{c}_{X,(2,j)}\}_{j=1}^{k_2}, ..., \{\vec{c}_{X,(l,j)}\}_{j=1}^{k_l}, \\ \{\vec{c}_{Y,(1,1)}\}, \{\vec{c}_{Y,(2,1)}\}, ..., \{\vec{c}_{Y,(h,1)}\}\}. \end{cases}$$

Définition 3.4.21 Considérons $C_X = \{\{\vec{c}_{X,(i,j)}\}_{j=1}^{k_i}\}_{i=1}^{l}$ et $C_Y = \{\{\vec{c}_{Y,(\iota,1)}\}\}_{\iota=1}^{h}$ telles que toutes les cellules de C_Y ont une ordonnée plus grande que les ordonnées des cellules de C_X. De plus, supposons que la cellule la plus haute de C_X a une arête commune

avec la cellule la plus basse de C_Y et que la colonne élémentaire simple de C_X ayant la position absolue la plus élevée est $\vec{c}_{X,(a,k_a)}$. Alors,

$$C_X +_{\underline{\uparrow}} C_Y = C_X\prime +_{\uparrow} \{\{\vec{c}_{Y,(2,1)}\}, \{\vec{c}_{Y,(3,1)}\}, ..., \{\vec{c}_{Y,(h,1)}\}\}$$

où $C_X\prime = \{\{\vec{c}_{X,(1,j)}\}_{j=1}^{k_1}, ..., \{\vec{c}_{X,(a,1)}, \vec{c}_{X,(a,2)}, ..., \vec{c}_{X,(a,k_a-1)}, \vec{c}_X\prime\}, ..., \{\vec{c}_{X,(l,j)}\}_{j=1}^{k_l}\}$

avec $\vec{c}_X\prime = (n_{X,(a,k_a)} + n_{Y,(1,1)}, m_{X,(a,k_a)})$.

Définition 3.4.22 Considérons $C_X = \{\{\vec{c}_{X,(i,j)}\}_{j=1}^{k_i}\}_{i=1}^{l}$ et $C_Y = \{\{\vec{c}_{Y,(\iota,1)}\}\}_{\iota=1}^{h}$. Alors,

$$C_X + C_Y = \begin{cases} C_X +_{\uparrow} C_Y \text{ sous la condition A,} \\ C_X +_{\underline{\uparrow}} C_Y \text{ sous la condition A',} \\ C_X +_{\updownarrow} C_Y \text{ sous les conditions B et 1,} \\ C_X +_{\underline{\updownarrow}} C_Y \text{ sous les conditions B et 2,} \\ C_X +_{\overline{\updownarrow}} C_Y \text{ sous les conditions B et 3,} \\ C_X +_{\underline{\overline{\updownarrow}}} C_Y \text{ sous les conditions B et 4,} \\ C_X +_{\circ} C_Y \text{ sous les conditions B et 5,} \\ C_X +_{\downarrow} C_Y \text{ sous la condition C,} \\ C_X +_{\overline{\downarrow}} C_Y \text{ sous la condition C'} \end{cases}$$

où les conditions sont définies comme suit.

- **Condition A :** Toutes les cellules de C_Y ont une ordonnée plus grande que celle de toute cellule de C_X et aucune cellule de C_Y n'a d'arête commune avec une cellule de C_X.
- **Condition A' :** Toutes les cellules de C_Y ont une ordonnée plus grande que celle de toute cellule de C_X et la cellule la plus basse de C_Y a une arête commune avec la cellule la plus haute de C_X.

- **Condition B :** $\vec{c}_{X,(a,b)}$ et $\vec{c}_{X,(\alpha,\beta)}$ sont voisines dans C_X et la position absolue de $\vec{c}_{X,(a,b)}$ est plus petite que celle de $\vec{c}_{X,(\alpha,\beta)}$. De plus, C_Y peut s'imbriquer dans C_X entre $\vec{c}_{X,(a,b)}$ et $\vec{c}_{X,(\alpha,\beta)}$.
- **Condition C :** Toutes les cellules de C_Y ont une ordonnée plus petite que celle de toute cellule de C_X et aucune cellule de C_Y n'a d'arête commune avec une cellule de C_X.
- **Condition C' :** Toutes les cellules de C_Y ont une ordonnée plus petite que celle de toute cellule de C_X et la cellule la plus haute de C_Y a une arête commune avec la cellule la plus basse de C_X.
- **Condition 1 :** Aucune cellule de C_X n'a d'arête commune avec toute cellule de C_Y.
- **Condition 2 :** La cellule la plus basse de C_Y a une arête commune avec la cellule la plus haute de $\vec{c}_{X,(a,b)}$ sans que la cellule la plus haute de C_Y n'ait une arête commune avec la cellule la plus basse de $\vec{c}_{X,(\alpha,\beta)}$.
- **Condition 3 :** La cellule la plus haute de C_Y a une arête commune avec la cellule la plus basse de $\vec{c}_{X,(\alpha,\beta)}$ sans que la cellule la plus basse de C_Y n'ait une arête commune avec la cellule la plus haute de $\vec{c}_{X,(a,b)}$.
- **Condition 4 :** la cellule la plus haute de C_Y a une arête commune avec la cellule la plus basse de $\vec{c}_{X,(\alpha,\beta)}$, la cellule la plus basse de C_Y a une arête commune avec la cellule la plus haute de $\vec{c}_{X,(a,b)}$ et $h > 1$.
- **Condition 5 :** la cellule la plus haute de C_Y a une arête commune avec la cellule la plus basse de $\vec{c}_{X,(\alpha,\beta)}$, la cellule la plus basse de C_Y a une arête commune avec la cellule la plus haute de $\vec{c}_{X,(a,b)}$ et $h = 1$.

Remarque 3.4.4 Si C_{I_1} et C_{I_2} sont deux colonnes complémentaires de C_0 associées à des vides élémentaires distincts de C_0, alors la colonne pouvant suivre C_0 qui est constituée de la précolonne C_X, de C_{I_1} et C_{I_2} est $(C_X + C_{I_1}) + C_{I_2}$. L'expression $C_X + (C_{I_1} + C_{I_2})$ n'est pas définie ici, alors on n'utilisera pas les parenthèses et les opérations seront effectuées de gauche à droite. △

Il est possible de définir en extension l'ensemble $\{C_s\}_0$ des colonnes pouvant être immédiatement à droite d'une colonne C_0 et d'écrire chaque C_s sous la forme $C_s = \{\{\vec{c}_{s,(i,j)}\}_{j=1}^{k_i}\}_{i=1}^{l}$. En effet, si $\{C_X\}$ est l'ensemble des précolonnes de C_0 et $\{\{C_{I_1}, C_{I_2}, ..., C_{I_N}\}_Y\}$ est l'ensemble des ensembles de colonnes complémentaires associées aux N vides élémentaires de C_0, alors on peut écrire chaque C_s en calculant

$$C_s = C_X + \sum_{i=1}^{N} C_{I_i}$$

pour chaque élément de $\{C_X\} \times \{\{C_{I_1}, C_{I_2}, ..., C_{I_N}\}_Y\}$.

3.5 Récurrence sur l'aire des polyominos

Cette section a pour but d'établir une relation de récurrence sur l'aire des polyominos à hauteur bornée. Ceux-ci seront énumérés et cela permettra de dénombrer les polyominos à forme fixée dont l'aire n est donnée.

Définition 3.5.1 Un polyomino s'obtient par la translation d'un autre si et seulement si ces polyominos sont le même ***polyomino à forme fixée***.

Théorème 3.5.1 *La colonne la plus à droite d'un polyomino n'a qu'une seule colonne élémentaire.*

Preuve Si la colonne la plus à droite d'un polyomino a plus d'une colonne élémentaire, alors ce polyomino n'est pas connexe, car par définition, ses colonnes élémentaires ne sont pas connectées.

■

Définition 3.5.2 Soit un polyomino P et une colonne C_0 de P qui n'est pas la plus à droite de P. Alors, l'ensemble des colonnes de P dont les cellules ont une abscisse plus grande que celle des cellules de C_0 forment la ***fin de*** P ***en partant de*** C_0.

Exemple 3.5.1 La figure suivante illustre, en gris pâle, une fin de polyomino en partant de la colonne dont les cellules sont noires.

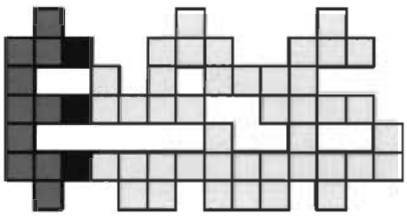

Figure 24 Une fin de polyomino en partant d'une colonne

Remarque 3.5.1 On s'intéressera aux fins possibles de polyominos à partir d'une colonne C_0. Cela correspond aux ensembles de cellules à droite de C_0 formant un polyomino avec C_0 et les colonnes à gauche de C_0. Par exemple, les fins possibles avec 4 cellules en partant de $\{\{(1,2)\},\{(1,4)\}\}$ sont à la figure suivante.

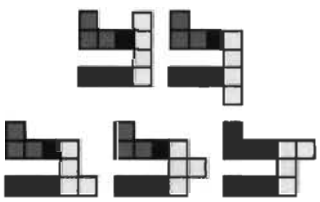

Figure 25 Les fins possibles

△

Remarque 3.5.2 Si n est l'aire des polyominos qu'on veut dénombrer, il suffit d'énumérer, pour chaque colonne C_0 pouvant être la colonne la plus à gauche d'un tel polyomino, les fins possibles avec $n_1 = n - |C_0|$ cellules à partir de C_0. De plus, pour

une colonne C_1 d'un polyomino quelconque, on peut énumérer les fins possibles avec n_1 cellules en partant de C_1. En effet, il suffit d'énumérer, pour chaque colonne C_2 pouvant être la colonne immédiatement à droite de C_1, les fins possibles avec $n_1 - |C_2|$ cellules en partant de C_2 et d'additionner les résultats. En posant $F(C, n)$ égale au nombre de fins possibles avec n cellules en partant de la colonne C, on obtient, si $n > 0$,

$$F(C, n) = \sum_{X \in S(C)} F(X, n - |X|)$$

où $S(C)$ est l'ensemble des colonnes pouvant être immédiatement à droite de C. Remarquez que $F(C, n)$ ne dépend que de ce qui peut être placé immédiatement à droite de C. Si $n = 0$, alors

$$F(C, n) = \begin{cases} 1 \text{ si } C \text{ est une colonne élémentaire,} \\ 0 \text{ sinon} \end{cases}$$

selon le théorème 3.5.1. De plus, $F(C, n) = 0$ si $n < 0$. △

Dans un contexte où les polyominos à énumérer sont à hauteur bornée, il est possible de déterminer $S(C)$ en utilisant la technique présentée à la fin de la section 3.4.

Pour énumérer les polyominos d'aire n généraux, on énumère d'abord les polyominos d'aire n dont la hauteur est bornée par n. Ensuite, pour tenir compte du fait que certains polyominos ainsi énumérés sont la translation d'un autre, on élimine tous ceux dont la hauteur est bornée par $n - 1$. On obtient ainsi le nombre de polyominos généraux d'aire n.

CHAPITRE 4

LES POLYOMINOS D'AIRE MINIMALE

4.1 Introduction

Dans ce chapitre, nous énumérerons de différentes manières l'ensemble $P_{min}(b, h)$ des polyominos d'aire minimale inscrits dans un rectangle $b \times h$. Nous verrons qu'un tel polyomino a une structure *équerre* · *escalier* · *équerre* et nous définirons ce que sont les équerres et les escaliers. Nous verrons une formule récursive, une formule directe et une série génératrice qui permettront d'énumérer ces polyominos minimaux. On construira la série génératrice des polyominos d'aire minimale en multipliant les séries des escaliers et des équerres. Un article de Alain Goupil, Hugo Cloutier et Fathallah Nouboud [7] porte sur ce sujet. La recherche visant à énumérer ces polyominos a été effectuée par Alain Goupil et Hugo Cloutier ; Fathallah Nouboud a contribué en développant quelques programmes pour tester les résultats et orienter les réflexions.

4.2 Définitions

Les équerres et les escaliers sont des sous-ensembles de cellules d'un polyomino d'aire minimale. Ils sont aussi des polyominos d'aire minimale.

Définition 4.2.1 Une ***équerre*** est un polyomino d'aire minimale (Voir la définition 1.1.4) où pour une cellule c de ce polyomino, il y a

- un ensemble de cellules non vide connexe sur la même ligne que c et du même côté de c et
- un ensemble de cellules non vide connexe dans la même colonne que c et dont les cellules sont toutes en haut de c ou toutes en bas de c.

La cellule c est appelée le coin de l'équerre.

Exemple 4.2.1 La figure suivante présente une équerre.

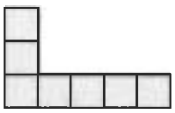

Figure 26 Une équerre

Définition 4.2.2 Un ***escalier*** est un polyomino d'aire minimale occupant deux coins diagonalement opposés d'un rectangle qui lui est circonscrit.

Exemple 4.2.2 La figure suivante illustre un escalier.

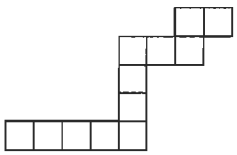

Figure 27 Un escalier

Définition 4.2.3 Un polyomino ***croix*** est un polyomino d'aire minimale construit d'une rangée et d'une colonne de cellules se croisant en une cellule. Une telle rangée ou colonne peut être constituée d'une seule cellule.

Exemple 4.2.3 On trouve un polyomino croix sur la figure suivante.

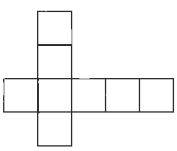

Figure 28 Un polyomino croix

4.3 Une description de l'ensemble des polyominos d'aire minimale

Nous considérerons l'ensemble des polyominos minimaux $P_{\downarrow}(b,h)$ pour lesquels :

1. La cellule la plus à gauche est plus haute ou à la même hauteur que la cellule la plus à droite.
2. La cellule la plus haute a une abscisse plus petite ou égale à celle de la cellule la plus basse.

Considérons aussi $P_{\uparrow}(b,h)$, l'ensemble des polyominos de $P_{min}(b,h)$ tels que :

1. La cellule la plus à gauche est plus basse ou à la même hauteur que la cellule la plus à droite.
2. La cellule la plus haute à une abscisse plus grande ou égale à celle de la cellule la plus basse.

Les polyominos de $P_{\uparrow}(b,h)$ sont comme celui de la figure 4. Ceux de $P_{\downarrow}(b,h)$ ont plutôt l'escalier (la partie entre les équerres) descendant de gauche à droite.

Disons aussi que $P_{+}(b,h)$ est l'ensemble des polyominos croix inscrits dans un rectangle $b \times h$. Ces polyominos croix sont à la fois dans $P_{\downarrow}(b,h)$ et $P_{\uparrow}(b,h)$. De plus, comme on peut établir une bijection entre $P_{\downarrow}(b,h)$ et $P_{\uparrow}(b,h)$, on a

$$|P_{min}(b,h)| = 2|P_{\downarrow}(b,h)| - |P_{+}(b,h)|. \tag{4.1}$$

4.4 Une récurrence

Soit $P_{1,1}(b,h)$, l'ensemble des polyominos de $P_{\downarrow}(b,h)$ qui ont une cellule c dans le coin supérieur gauche d'un rectangle $b \times h$. Alors, si $b,h > 1$ (si $b = 1$ ou $h = 1$, alors $|P_{1,1}(b,h)| = 1$), soit il y a une cellule immédiatement à droite de c sans qu'il n'y ait de cellule immédiatement sous c, soit il y a une cellule immédiatement sous c sans qu'il n'y ait de cellule immédiatement à droite de c, soit il y a une cellule immédiatement à droite de c et une immédiatement sous c. Ainsi, dans les deux premiers cas, en enlevant

c, on obtient des polyominos d'aire minimale inscrits dans un rectangle pour lequel la base est plus petite que b ou la hauteur est plus petite que h. Dans le troisième cas, il n'y a qu'une construction possible qui suit la contrainte d'aire minimale : l'équerre. Alors,

$$|P_{1,1}(b,h)| = \begin{cases} |P_{1,1}(b,h-1)| + |P_{1,1}(b-1,h)| + 1 \text{ si } b,h \geq 1, \\ 0 \text{ si } b = 0 \text{ ou } h = 0. \end{cases}$$

Remarque 4.4.1 Les polyominos de $P_{1,1}(b,h)$ seront appelés polyominos coins. Il y a une bijection entre l'ensemble des escaliers allant du coin supérieur gauche à tout point à coordonnées entières d'un rectangle et les polyominos coins inscrits dans ce rectangle. En effet, le choix d'un tel escalier impose une seule manière de choisir une équerre pour compléter la construction. Aussi, la formule sommatoire énumérant ces escaliers allant du coin supérieur gauche à tout point à coordonnées entières d'un rectangle se simplifie comme suit :

$$|P_{1,1}(b,h)| = \sum_{i=1}^{h-1}\sum_{j=1}^{b-1}\binom{i+j-2}{i-1} + \binom{h+b-2}{h-1} \quad = 2\binom{b+h-2}{b-1} - 1.$$

De plus, par l'identité du binôme de Newton, on trouve la série génératrice

$$\sum_{b,h\geq 1} |P_{1,1}(b,h)| x^b y^h = \underbrace{\left(1 + \frac{xy}{(1-x)(1-y)}\right)\frac{xy}{1-x-y}}_{équerre \times escalier}.$$

△

Considérons aussi l'ensemble des polyominos de $P_{\downarrow}(b,h)$ qui n'ont pas de cellule dans le coin supérieur gauche d'un rectangle $b \times h$. Alors, pour un polyomino p de cet ensemble, si on se place sur la cellule la plus à gauche de p puis qu'on se déplace vers la droite de cellule en cellule, il y aura, sur ce parcourt, une cellule c_x au dessus de laquelle se trouve une cellule. S'il y a $i-1$ cellules au dessus de c_x et $j-1$ cellules à gauche de c_x $(i,j>1)$(Ces $i-1$ cellules et ces $j-1$ cellules forment une équerre avec

c_x qui en est le coin.), alors p est dans le sous-ensemble $P_{i,j}(b,h)$ de $P_{\downarrow}(b,h)$ et

$$|P_{i,j}(b,h)| = |P_{1,1}(b-j+1, h-i+1)|.$$

Il est alors possible d'évaluer $|P_{\downarrow}(b,h)|$, car

$$|P_{\downarrow}(b,h)| = |P_{1,1}(b,h)| + \sum_{i=2}^{h}\sum_{j=2}^{b} |P_{i,j}(b,h)|.$$

Et, $|P_{+}(b,h)| = bh$, donc

$$|P_{min}(b,h)| = \begin{cases} 2(|P_{1,1}(b,h)| + \sum_{i=2}^{h}\sum_{j=2}^{b}|P_{i,j}(b,h)|) - bh \text{ si } b,h > 1 \\ 1 \text{ si } b = 1 \text{ ou } h = 1. \end{cases}$$

où $|P_{1,1}(b,h)|$ et $|P_{i,j}(b,h)|$ peuvent être calculés de façon récursive.

4.5 Une formule directe

Voyons maintenant une formule non récursive pour calculer $|P_{\downarrow}(b,h)|$. Soit

$$P_{escalier}(i,j),$$

le nombre de polyominos d'aire minimale inscrits dans un rectangle $i \times j$ avec une cellule dans le coin supérieur gauche et une autre dans le coin inférieur droit. Alors,

$$|P_{escalier}(i,j)| = \begin{pmatrix} i+j-2 \\ i-1 \end{pmatrix}. \tag{4.2}$$

En effet, on peut voir un élément de $P_{escalier}$ comme un chemin constitué de pas vers la droite et de pas vers le bas. Il y a $i+j-1$ cellules à un tel polyomino, donc le chemin correspondant a $i+j-2$ pas. Pour déterminer un tel chemin, on choisi $i-1$ pas vers la droite parmi $i+j-2$ pas.

Les polyominos de $P_{1,1}(b,h)$ sont en correspondance biunivoque avec ceux de $\{P_{escalier}(i,j)|i<h, j<b\} \cup P_{escalier}(b,h)$. Donc,

$$|P_{1,1}(b,h)| = \sum_{i=1}^{h-1}\sum_{j=1}^{b-1}\begin{pmatrix} i+j-2 \\ i-1 \end{pmatrix} + \begin{pmatrix} h+b-2 \\ h-1 \end{pmatrix},$$

$$|P_{i,j}(b,h)| = \sum_{k=1}^{h-i}\sum_{l=1}^{b-j}\begin{pmatrix} k+l-2 \\ k-1 \end{pmatrix} + \begin{pmatrix} h-i+b-j \\ h-i \end{pmatrix} \text{ et}$$

$$|P_{min}(b,h)| = \begin{cases} 2\left[\sum_{i=1}^{h-1}\sum_{j=1}^{b-1}\begin{pmatrix} i+j-2 \\ i-1 \end{pmatrix} + \begin{pmatrix} h+b-2 \\ h-1 \end{pmatrix}\right. \\ \quad \left. + \sum_{i=2}^{h}\sum_{j=2}^{b}\left[\sum_{k=1}^{h-i}\sum_{l=1}^{b-j}\begin{pmatrix} k+l-2 \\ k-1 \end{pmatrix} + \begin{pmatrix} h-i+b-j \\ h-i \end{pmatrix}\right]\right] \\ \quad -bh \text{ si } b,h>1, \\ 1 \text{ si } b=1 \text{ ou } h=1. \end{cases}$$

L'identité $\sum_{j=k}^{n}\begin{pmatrix} j \\ k \end{pmatrix} = \begin{pmatrix} n+1 \\ k+1 \end{pmatrix}$ permet d'obtenir

$$p_{min}(b,h) = |P_{min}(b,h)| = \begin{cases} 8\begin{pmatrix} h+b-2 \\ h-1 \end{pmatrix} - 6 - bh - 2(b-1)(h-1) \text{ si } b,h>1, \\ 1 \text{ si } b=1 \text{ ou } h=1. \end{cases}$$

4.6 **Une série génératrice**

Voyons premièrement la série génératrice associée aux équerres sans leur cellule du coin. Il y a un seul segment horizontal d'une longueur donnée. La série génératrice pour ces segments est

$$\sum_{b\geq 1} x^b = \frac{x}{1-x},$$

où on tient compte du fait qu'il faut au moins une cellule pour qu'il y ait un segment. La série génératrice des segments verticaux se trouve de cette même manière et donc, pour les équerres sans coin $e(b,h)$ qui sont composées d'un segment vertical de longueur h et d'un segment horizontal de longueur b,

$$\sum_{b,h\geq 1} e(b,h)x^b y^h = \sum_{b\geq 1} x^b \cdot \sum_{h\geq 1} y^h = \frac{xy}{(1-x)(1-y)}$$

Voyons maintenant la série génératrice des escaliers. Par l'équation (4.2),

$$\begin{aligned}
\sum_{b,h\geq 1} |P_{escalier}(b,h)|x^b y^h &= \sum_{b,h\geq 1} \binom{b+h-2}{b-1} x^b y^h \\
&= xy \sum_{b,h\geq 0} \binom{b+h}{b} x^b y^h \\
&= xy \sum_{h+b\geq 0} \sum_{b\geq 0} \binom{h}{b} x^b y^{h-b} \\
&= xy \sum_{h\geq b} \sum_{b\geq 0} \binom{h}{b} x^b y^{h-b} \\
&= xy \sum_{h\geq 0} \sum_{b=0}^{h} \binom{h}{b} x^b y^{h-b} \\
&= xy \sum_{h\geq 0} (x+y)^h \\
&= \frac{xy}{1-x-y}
\end{aligned}$$

En réunissant les séries génératrices des équerres et des escaliers et en tenant compte de la structure *équerre* · *escalier* · *équerre* des polyominos minimaux où il est possible de ne pas choisir d'équerre, nous avons que

$$\sum_{b,h\geq 1} |P_{\downarrow}(b,h)|x^b y^h = \left(1 + \frac{xy}{(1-x)(1-y)}\right)^2 \frac{xy}{1-x-y}. \tag{4.3}$$

Il ne manque plus que la série génératrice des polyominos de $P_+(b,h)$. Pour un rectangle non dégénéré $b \times h$, $|P_+(b,h)| = bh$. La série génératrice associée à $P_+(b,h)$ est donc

$$\begin{aligned}
\sum_{b,h\geq 2} |P_+(b,h)|x^b y^h &= \sum_{b,h\geq 2} bhx^b y^h \\
&= xy\frac{d}{dx}\frac{d}{dy}\sum_{b,h\geq 2} x^b y^h \\
&= xy\frac{d}{dx}\frac{d}{dy}\left(\sum_{b\geq 0} x^b - 1 - x\right)\left(\sum_{h\geq 0} y^h - 1 - y\right) \\
&= xy\frac{d}{dx}\left(\sum_{b\geq 0} x^b - 1 - x\right)\left(\frac{1}{(1-y)^2} - 1\right) \\
&= xy\left(\frac{1}{(1-y)^2} - 1\right)\left(\frac{1}{(1-x)^2} - 1\right)
\end{aligned}$$

Si le rectangle $b \times h$ est dégénéré ($b = 1$ ou $h = 1$), alors la série génératrice est

$$xy\left(\sum_{b\geq 1} x^b + \sum_{h\geq 1} y^h + 1\right) = xy\left(\frac{1}{1-x} + \frac{1}{1-y} - 1\right).$$

Donc,

$$\sum_{b,h\geq 1} |P_+(b,h)|x^b y^h = xy\left[\left(\frac{1}{(1-y)^2} - 1\right)\left(\frac{1}{(1-x)^2} - 1\right) + \left(\frac{1}{1-x} + \frac{1}{1-y} - 1\right)\right]. \tag{4.4}$$

Alors, par les équations (4.1), (4.3) et (4.4),

$$\sum_{b,h\geq 1} |P_{min}(b,h)|x^b y^h = 2\left(1+\frac{xy}{(1-x)(1-y)}\right)^2 \frac{xy}{1-x-y} - xy\left[\left(\frac{1}{(1-y)^2}-1\right)\left(\frac{1}{(1-x)^2}-1\right)+\left(\frac{1}{1-x}+\frac{1}{1-y}-1\right)\right].$$

Voici un tableau donnant le nombre de polyominos d'aire minimale inscrits dans un rectangle de dimensions $b \times h$.

b/h	2	3	4	5	6	7	8	9	10
2	4	8	12	16	20	24	28	32	36
3	8	25	50	83	124	173	230	295	368
4	12	50	120	230	388	602	880	1230	1660
5	16	83	230	497	932	1591	2538	3845	5592
6	20	124	388	932	1924	3588	6212	10156	15860
7	24	173	602	1591	3588	7265	13582	23859	39856
8	28	230	880	2538	6212	13582	27288	51290	91308
9	32	295	1230	3845	10156	23859	51290	102745	194240
10	36	368	1660	5592	15860	39856	91308	194240	388692

Tableau I Nombre de polyominos d'aire minimale dans un rectangle

Nous pouvons aussi nous intéresser à $p_{min}(n)$, le nombre de polyominos d'aire minimale égale à n. Considérant que $n = b + h - 1$, nous avons

$$p_{min}(n) = \sum_{b=1}^{n} p_{min}(b,h) = 2^{n+2} - \frac{1}{2}(n^3 - n^2 + 10n + 4).$$

Le tableau suivant donne les premières valeurs de $p_{min}(n)$.

n	1	2	3	4	5	6	7	8	9	10
$p_{min}(n)$	1	2	6	18	51	134	328	758	1677	3594

Tableau II Nombre de polyominos d'aire minimale n

CHAPITRE 5

LES POLYOMINOS DE VOLUME MINIMAL

5.1 Introduction

Dans les pages qui suivent, nous faisons l'énumération des polyominos de volume minimal par des séries génératrices. Aussi, pour les polyominos de volume minimal d'un volume fixé, une formule directe est présentée. Un article de Alain Goupil et Hugo Cloutier [8] porte sur ce sujet.

Dans le but de vérifier numériquement certains résultats de ce chapitre, des programmes ont été exécutés sur différents ordinateurs, notamment sur ceux du CLUMEQ, un organisme chargé d'offrir du temps d'utilisation du système informatique le plus puissant de l'est du Canada. Pour énumérer ces polyominos, des formules sommatoires facilement programmables se trouvent à l'annexe 2 et un autre programme est à l'annexe 3.

5.2 Définitions

Définition 5.2.1 Un polyomino tridimensionnel est ***inscrit*** dans un prisme rectangulaire si et seulement s'il est en contact avec chaque face de ce prisme.

Pour construire un polyomino inscrit dans un prisme rectangulaire de profondeur b, de hauteur h et de longueur k, il faut au moins b cellules pour relier la face du devant à celle du derrière, h cellules pour relier la face du haut à celle du bas et k cellules pour relier la face gauche à la face droite. De plus, il y a une cellule qui contribue dans chacune de ces directions. Ainsi, $b + k + h - 2$ cellules suffisent pour construire un polyomino inscrit dans un prisme de dimensions $b \times k \times h$.

Définition 5.2.2 Un polyomino tridimensionnel est ***de volume minimal*** si et seulement s'il est constitué d'exactement $b + k + h - 2$ cellules lorsqu'on l'inscrit dans un prisme de dimensions $b \times k \times h$.

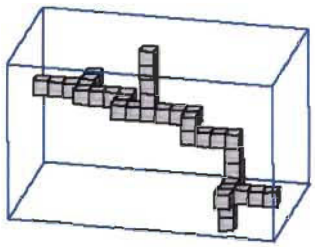

Figure 29 Un polyomino de volume minimal

5.3 Les polyominos diagonaux

Cette section traite de polyominos tridimensionnels de volume minimal ayant une structure analogue à celle des polyominos d'aire minimale, c'est-à-dire la structure *équerre* × *escalier* × *équerre*. Nous appellerons polyominos diagonaux ces polyominos de volume minimal particuliers. Pour les construire, nous devons d'abord définir ce que sont un escalier 3D et une équerre 3D. Aussi, les équerres qui seront placées aux bouts d'un escalier pourront se présenter sous la forme d'une équerre 3D, d'une équerre 2D ou d'un tripode. Nous définirons ces différents objets dans les prochaines pages.

Définition 5.3.1 Le ***degré*** d'une cellule est le nombre de cellules en contact avec celle-ci.

Définition 5.3.2 Un ***pilier*** est un polyomino 3D dont les cellules sont placées sur une droite. De plus, un pilier relie une face du prisme circonscrit au reste du polyomino.

Définition 5.3.3 Une ***équerre 3D*** est un polyomino de volume minimal constitué d'un polyomino coin 2D et d'un ensemble connexe non vide de cellules sur une demi-

droite orthogonale au plan du polyomino coin. Cette demi-droite débute à la cellule du coin.

À la figure 30, on voit une équerre 3D. Son polyomino coin est en bleu avec la cellule du coin en rouge. En vert, on trouve l'ensemble de cellules sur la demi-droite orthogonale au plan contenant le polyomino coin. Cette partie verte sera aussi appelée pilier.

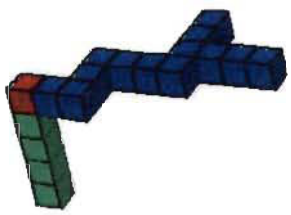

Figure 30 Une équerre 3D

Construisons une série génératrice associée aux équerres 3D. Pour ce faire, il faut d'abord trouver la série génératrice des polyominos coins inscrits dans un rectangle $b \times h$. Disons que ces polyominos sont ceux de l'ensemble $P_{1,1}(b,h)$. Voyant un polyomino coin comme un escalier auquel on ajoute possiblement une équerre et utilisant l'équation 4.3, nous trouvons

$$\begin{aligned} \sum_{b,h\geq 1} |P_{1,1}(b,h)| x^b y^h &= \left(1 + \frac{xy}{(1-x)(1-y)}\right) \frac{xy}{1-x-y} \\ &= \frac{2xy}{1-x-y} - \frac{xy}{(1-x)(1-y)}. \end{aligned} \tag{5.1}$$

Pour construire la série génératrice $\acute{E}querre3D(x,y,z)$ des équerres 3D, il suffit de reprendre celle des polyominos coins pour chaque plan et de lui ajouter la série génératrice d'un pilier ayant au moins une cellule et placé sur une droite orthogonale à ce plan. Il faut aussi enlever les équerres (polyominos coins sans escalier) de l'ensemble

des polyomino coins, car autrement, nous construirions des tripodes. Ceux-ci seront énumérés par une autre série génératrice. La série génératrice des équerres 3D moins les tripodes est donc

$$\begin{aligned} \acute{E}querre3D(x,y,z) = &\left[\frac{2yz}{1-y-z} - \frac{2yz}{(1-y)(1-z)}\right]\frac{x^2}{(1-x)} \\ &+ \left[\frac{2xz}{1-x-z} - \frac{2xz}{(1-x)(1-z)}\right]\frac{y^2}{(1-y)} \\ &+ \left[\frac{2xy}{1-x-y} - \frac{2xy}{(1-x)(1-y)}\right]\frac{z^2}{(1-z)}. \end{aligned}$$

Une équerre 3D dont le polyomino coin est une équerre (dans ce cas le polyomino coin n'a pas d'escalier) est un tripode. Un tripode est constitué de trois piliers orthogonaux se joignant en une cellule. Dans la série génératrice des tripodes, on utilisera une variable distincte pour chaque pilier. Ces piliers ont chacun au moins 2 cellules. Si nous désignons la série génératrice des tripodes par $Tripode(x,y,z)$, alors

$$Tripode(x,y,z) = \sum_{i,j,k\geq 2} x^i y^j z^k = \frac{x^2y^2z^2}{(1-x)(1-y)(1-z)}.$$

On peut enlever le pilier et l'escalier d'une équerre 3D. Il reste alors une équerre dans un plan. Nous appellerons la série génératrice de ces équerres $\acute{E}querre2D(x,y,z)$qui sont au moins de format 2×2. Considérant que ces équerres peuvent être dans n'importe quel des trois plans, leur série génératrice est

$$\acute{E}querre2D(x,y,z) = \frac{x^2y^2z}{(1-x)(1-y)} + \frac{x^2yz^2}{(1-x)(1-z)} + \frac{xy^2z^2}{(1-z)(1-y)}.$$

Un escalier 3D est un polyomino correspondant à un chemin de longueur minimale reliant deux coins diagonalement opposés d'un prisme. Si nous nommons $Escalier3D(x,y,z)$ la série génératrice des escaliers 3D, alors

$$\begin{aligned} Escalier3D(x,y,z) &= \sum_{i,j,k\geq 1} \begin{pmatrix} i+j+k-3 \\ i-1, j-1, k-1 \end{pmatrix} x^i y^j z^k \\ &= xyz \sum_{n\geq 0} (x+y+z)^n \\ &= \frac{xyz}{1-x-y-z} \end{aligned}$$

Les polyominos diagonaux sont formés d'un escalier 3D ayant possiblement à chaque bout un tripode, une équerre 2D ou une équerre 3D, mais cette construction se fait en suivant une diagonale déterminée par le choix des coins du prisme dans lesquels seront placées les équerres. Si nous désignons par $1Diag(x,y,z)$ la série génératrice des polyominos diagonaux placés le long d'une diagonale donnée, alors

$$\begin{aligned} 1Diag(x,y,z) &= Escalier3D(x,y,z) \\ &\cdot \left[1 + \frac{Tripode(x,y,z) + Équerre3D(x,y,z) + Équerre2D(x,y,z)}{xyz}\right]^2. \end{aligned}$$

Le prisme dans lequel sont inscrits les polyominos diagonaux contient 4 diagonales au long desquelles nous pouvons construire ces polyominos. Toutefois, nous verrons qu'il est possible de construire des polyominos diagonaux qui suivront 2, 3 ou 4 diagonales à la fois. La figure 31 présente un polyomino pouvant être construit sur au moins 2 diagonales. Un tel polyomino est généralement constitué de deux polyominos coins (en bleu sur la figure 31) placés dans un même plan et se joignant en une cellule (leur cellule de coin). À partir de cette cellule, deux piliers (en vert sur la figure 31) sont construits sur la droite orthogonale au plan contenant les polyominos coins et passant par la cellule où se joignent ces polyominos coins.

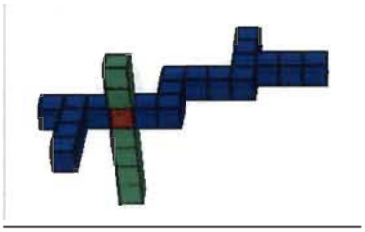

Figure 31 Un polyomino de volume minimal construit sur 2 diagonales

Si nous désignons par $2Diag_z(x, y, z)$ la série génératrice des polyominos diagonaux pouvant être construits sur au moins 2 diagonales avec les polyominos coins (Voir l'équation 5.1.) dans un plan parallèle au plan XY (plan contenant l'abscisse et l'ordonnée) et si nous considérons que ces polyominos coins peuvent être placés de deux manières (Ils forment un polyomino d'aire minimale et celui-ci peut avoir un escalier ascendant ou descendant), alors

$$2Diag_z(x, y, z) = 2 \times \frac{1}{xy}\left(\frac{2xy}{1-x-y} - \frac{xy}{(1-x)(1-y)}\right)^2 z\left(\frac{1}{1-z}\right)^2.$$

Il y a aussi des diagonaux pouvant être construits sur 3 diagonales, mais dans ce cas, on peut se convaincre qu'ils sont forcément sur 4 diagonales. Ces polyominos sont constitués strictement de piliers orthogonaux se joignant en une cellule comme le montre la figure 32.

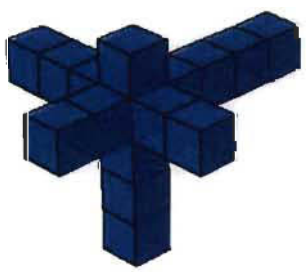

Figure 32 Un polyomino de volume minimal construit sur 4 diagonales

Pour simplifier la série génératrice des polyominos qui sont sur 4 diagonales, nous ne considérerons que le cas où le prisme dans lequel sont inscrits ces polyominos est non dégénéré, c'est-à-dire que ses arêtes sont au moins de longueur 2. Le cas où ce prisme est dégénéré revient à énumérer des polyominos d'aire minimale et cela a été fait au chapitre 4. Sachant qu'il y en a bhk (le nombre de manières de choisir le point d'intersection des piliers) dans un prisme de dimension $b \times h \times k$ avec $b, h, k \geq 2$, si $4Diag(x, y, z)$ est la série génératrice des polyominos construits sur 4 diagonales, alors

$$4Diag(x, y, z) = \sum_{b,k,h \geq 2} bkhx^b y^k z^h = \frac{x^2(2-x)y^2(2-y)z^2(2-z)}{(1-x)^2(1-y)^2(1-z)^2}.$$

Nous utiliserons le principe d'inclusion exclusion pour énumérer les polyominos diagonaux. Dans un prisme rectangulaire, les polyominos placés sur 4 diagonales sont comptés 4 fois lorsque nous multiplions par 4 ceux qui sont sur au moins une diagonale (ceux de la série $1Diag(x, y, z)$) et 6 (le choix de 2 diagonales parmi 4) fois lorsque nous énumérons ceux qui sont sur au moins 2 diagonales. Ainsi, en énumérant ceux qui sont sur au moins une diagonale ($4 \times 1Diag(x, y, z)$) puis en enlevant tous ceux qui sont sur au moins deux diagonales (ces derniers ont été comptés 2 fois à la première étape), nous enlevons 2 fois ceux qui sont sur 4 diagonales. Pour compléter l'énumération des diagonaux, il reste à ajouter 3 fois ceux qui sont sur 4 diagonales. Ainsi, si $Diag(x, y, z)$ désigne la série génératrice des diagonaux inscrits dans un prisme rectangulaire, alors

$$\begin{aligned} Diag(x, y, z) = {} & 4 \times 1Diag(x, y, z) - (2Diag_x(x, y, z) + 2Diag_y(x, y, z) + 2Diag_z(x, y, z)) \\ & + 3 \times 4Diag(x, y, z) \end{aligned}$$

5.4 Polyominos non diagonaux

Les programmes des annexes 2 et 3 ont permis de détecter l'existence de polyominos de volume minimal qui ne sont pas diagonaux. Ces polyominos seront dits non diagonaux.

L'ensemble des polyominos non diagonaux se partitionne en deux classes : l'ensemble des polyominos 2DX2D et l'ensemble des croix gauches. À la section 5.5, il y a la preuve que tout polyomino de volume minimal est soit un diagonal, soit un 2DX2D, soit une croix gauche. Ainsi, lorsque nous aurons une série génératrice pour les 2DX2D et une pour les croix gauches, il ne restera qu'à additionner les séries génératrices des diagonaux, des 2DX2D et des croix gauches pour obtenir celle de tous les polyominos de volume minimal.

Définition 5.4.1 Un ***polyomino 2DX2D*** est constitué de deux polyominos d'aire minimale placés dans des plans orthogonaux avec en commun un bras d'équerre. En se référant à la figure 33, un polyomino 2DX2D doit avoir, en contact avec la cellule rouge de gauche, au moins une cellule devant et une cellule derrière. Aussi, en contact avec la cellule rouge de droite, il doit y avoir au moins une cellule en haut et une cellule en bas. La partie verte entre les deux cellules rouges peut être dégénérée. C'est-à-dire que les cellules rouges peuvent être en contact.

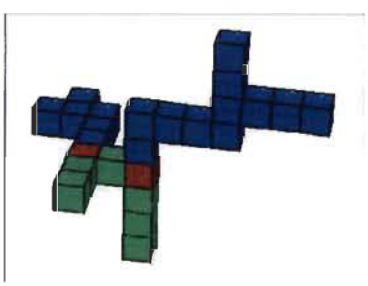

Figure 33 Un polyomino 2DX2D

Remarque 5.4.1 Les parties vertes (sauf celle entre les cellules rouges) et bleues doivent avoir au moins une cellule, sinon le polyomino devient diagonal. Aussi, les parties bleues (incluant les cellules rouges) sont des polyominos coins 2D et les parties vertes sont des pilier. △

Construisons la série génératrice des polyominos 2DX2D. Premièrement, observons le polyomino 2DX2D de la figure 33 et disons que la partie bleue verticale (à droite) est dans le plan YZ et que la partie bleue horizontale (à gauche) est dans le plan XY. Dans ce cas, si nous ne tenons pas compte de la contribution en x et en z du segment central (en vert), ni de celle en y des cellules rouges, la série génératrice $C(x, y, z)$ du centre fait de cellules vertes est

$$C(x, y, z) = \frac{x}{1-x} \times \frac{1}{1-y} \times \frac{z}{1-z}.$$

Pour ce qui est de la partie bleue verticale (à droite), c'est un polyomino coin dont la hauteur en z est au moins de 2. De l'équation 5.1, la série génératrice de ces polyominos coins est

$$Coin_{z\geq 2}(y, z) = yz\left[\frac{2}{1-y-z} - \frac{1}{(1-y)(1-z)} - \frac{1}{1-y}\right].$$

De manière similaire, pour la partie bleue horizontale (à gauche), nous avons

$$Coin_{x\geq 2}(x, y) = xy\left[\frac{2}{1-x-y} - \frac{1}{(1-x)(1-y)} - \frac{1}{1-y}\right].$$

Aussi, si nous fixons ce qui est dans le plan YZ, le pilier vert horizontal dans le plan XY peut être placé de deux manières : rejoignant la face de devant ou celle de derrière du prisme circonscrit. Pour tenir compte de cela, nous multiplierons par 2 la série $Coin_{x\geq 2}(x, y)$. Toutefois, certaines de ces constructions ont un pilier qui touche la face

du devant du prisme et un autre pilier qui touche celle du derrière. Ces constructions sont comptées 2 fois dans la série $2 \times Coin_{x\geq 2}(x,y)$; il faudra les soustraire une fois. Nous pouvons faire ces mêmes observations à propos de la partie qui est dans le plan YZ. De plus, il y a deux manières de placer la partie verticale dans le plan YZ. Elle peut être placée à gauche ou à droite. Selon ces considérations, la série génératrices des polyominos 2DX2D placés dans les plans XY et YZ est

$$P_{xy\times yz}(x,y,z) = 2\left[2 \times Coin_{x\geq 2}(x,y) - \frac{x^2 y}{(1-x)(1-y)}\right] \times C(x,y,z) \times \left[2 \times Coin_{z\geq 2}(y,z) - \frac{yz^2}{(1-y)(1-z)}\right].$$

Enfin, un ensemble de polyominos 2DX2D construits selon une paire de plans est disjoint d'un ensemble de polyominos 2DX2D construits selon une autre paire de plans. Considérant toutes les paires de plans possibles, la série génératrice des polyominos 2DX2D est

$$P_{2DX2D}(x,y,z) = P_{xy\times yz}(x,y,z) + P_{xy\times xz}(x,y,z) + P_{xz\times yz}(x,y,z).$$

Définition 5.4.2 Une ***croix gauche*** est constituée de trois polyominos coins deux à deux orthogonaux et se joignant en une cellule centrale. Cette cellule est le coin fixé de chacun de ces polyominos coins et ceux-ci sont inscrits dans des rectangles dont la largeur est au moins égale à 2. On distingue deux types de croix gauches :

- **Type A :** Deux faces parallèles de la cellule centrale sont en contact avec une autre cellule.
- **Type B :** Les trois faces de contact de la cellule centrale sont incidentes à un sommet de cette cellule.

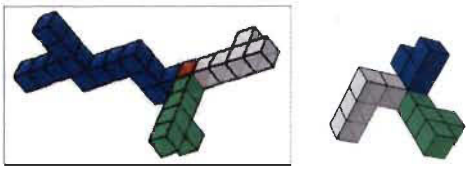

Figure 34 Des croix gauches de types A (à gauche) et B (à droite)

Construisons la série génératrice $CG_A(x, y, z)$ des croix gauches de type A. Considérons d'abord le cas où les faces de contact de la cellule centrale sont choisies et disons que les polyominos coins sont placés comme sur la figure 34 à gauche. Alors, en excluant la cellule rouge, disons que le polyomino coin bleu est dans le plan XZ, le vert dans le plan YZ et le gris dans le plan XY. Selon la définition 2, le polyomino coin bleu et rouge est inscrit dans un rectangle dont la largeur est au moins égale à 2. Donc, la série génératrice du polyomino coin bleu (sans la cellule rouge) est égale à celle des polyominos coins dans le plan XZ moins la série génératrice des polyominos coins inscrits dans un rectangle dont la hauteur (en z) est égale à 1. Ainsi, la série génératrice du polyomino coin bleu (sans la cellule rouge) est

$$PC_{z\geq 2}(x, z) = xz\left[\frac{2}{1-x-z} - \frac{1}{(1-x)(1-z)} - \frac{1}{1-x}\right].$$

De manière similaire, pour les polyominos coins vert et gris, nous avons

$$PC_{z\geq 2}(y, z) = yz\left[\frac{2}{1-y-z} - \frac{1}{(1-y)(1-z)} - \frac{1}{1-y}\right]$$

et

$$PC_{y\geq 2}(x, y) = xy\left[\frac{2}{1-x-y} - \frac{1}{(1-x)(1-y)} - \frac{1}{1-x}\right].$$

Pour ce qui est de la cellule rouge, le produit des trois dernières séries génératrices compte deux fois sa contribution en z, une fois celle en y et aucune fois celle en x. De plus, si on place ces polyominos coins un à la fois, il y a deux manières de placer en premier le polyomino coin vert dans le plan YZ : soit vers le haut, soit vers le bas. Ensuite, le polyomino coin bleu peut être placé soit dans le plan XZ, soit dans le plan XY. Le plan du troisième polyomino coin est alors déterminé et cela fait 4 manières de placer les polyominos coins. Ainsi, pour des faces de contact de la cellule rouge choisies pour former une croix gauche de type A, la série génératrice est

$$\begin{aligned} CG_{A1}(x,y,z) &= \frac{4x}{z} PC_{z\geq 2}(x,z) PC_{z\geq 2}(y,z) PC_{y\geq 2}(x,y) \\ &= \frac{4x^3y^3z^3(1+x-z)(1+y-z)(1+x-y)}{(1-x)^2(1-y)^2(1-z)^2(1-x-z)(1-y-z)(1-x-y)}. \end{aligned}$$

Pour les 12 autres façons de choisir des faces de contact de manière à former une croix gauche de type A, nous pouvons faire le même raisonnement. En faisant la somme de ces 12 résultats, nous obtenons la série génératrice des croix gauches de type A qui est

$$GC_A(x,y,z) = \frac{-16x^3y^3z^3[(1-x+y)(1-x+z)+(1-y+x)(1-y+z)+(1-z+x)(1-z+y)]}{(1-x)^2(1-y)^2(1-z)^2(1-y-z)(1-x-y)(1-x-z)}.$$

La série génératrice des croix gauches de type B, $CG_B(x,y,z)$, s'obtient en considérant que la cellule centrale a 8 sommets (déterminant les 3 faces de contact) et qu'une fois qu'un sommet est choisi, il y a deux façons de placer les polyominos coins. En effet, en se référant à la figure 34 à droite, si la face de contact de la cellule centrale avec le polyomino bleu est parallèle au plan XY, alors le polyomino bleu est dans un plan qui contient l'axe Z. Cela donne deux possibilités pour le plan du polyomino bleu soit XZ et YZ. Le choix d'une de ces possibilités détermine les plans des deux autres polyominos coins. Ces considérations étant faites,

$$CG_B(x,y,z) = 8(PC_{y\geq 2}(y,z)PC_{z\geq 2}(x,z)PC_{x\geq 2}(x,y) + PC_{x\geq 2}(x,z)PC_{y\geq 2}(x,y)PC_{z\geq 2}(y,z))$$
$$= \frac{16x^3y^3z^3((1-x+y)(1-x+z)+(1-y+x)(1-y+z)+(1-z+x)(1-z+y)-4)}{(1-x)^2(1-y)^2(1-z)^2(1-y-z)(1-x-y)(1-x-z)}.$$

La série génératrice des croix gauches, $CG(x,y,z)$, est la somme de la série génératrice des croix gauches de type A et de celle des croix gauches de type B. Après simplification, nous avons

$$CG(x,y,z) = \frac{64x^3y^3z^3}{(1-x-y)(1-x-z)(1-y-z)(1-x)^2(1-y)^2(1-z)^2}.$$

La forme rationelle de $CG(x,y,z)$ est beaucoup plus simple que les séries des croix gauches de type A et de type B. Toutefois, aucune formule directe n'en a été tirée. À la fin de ce chapitre se trouve une récurrence linéaire provenant de cette série.

5.5 Résultat principal

Dans cette section, nous prouverons que tout polyomino de volume minimal est soit un diagonal, soit un 2DX2D, soit une croix gauche. De plus, comme ces trois ensembles de polyominos sont disjoints, il suffira de faire la somme de leurs séries génératrices pour trouver celle de tous les polyominos de volume minimal.

Définition 5.5.1 Dans un polyomino de volume minimal, une ***cellule de contact*** touche la face du haut ou la face du bas du prisme dans lequel est inscrit ce polyomino. De plus, si cette cellule de contact touche la face du bas du prisme et que ses coordonnées sont (x_0, y_0, z_0), alors une cellule de ce polyomino a les coordonnées (x_0, y_0, z_0+1). De manière similaire, si cette cellule de contact touche la face du haut du prisme et que

ses coordonnées sont (x_1, y_1, z_1), alors une cellule de ce polyomino a les coordonnées $(x_1, y_1, z_1 - 1)$.

Remarque 5.5.1 Soit un polyomino 2D ou 3D P. Alors, P est d'aire ou de volume minimal si et seulement si P est connexe et que dans P, à partir de deux cellules distinctes a et b prises dans une même rangée ou colonne, pour un pas à partir de a orthogonal à cette rangée, il n'y a pas de pas à partir de b dans ce même sens. Autrement, des pas seraient fait deux fois pour une même contribution. △

Lemme 5.5.1 *Si un polyomino est de volume minimal, alors la projection sur une face de son prisme circonscrit est un polyomino d'aire minimale*

Preuve Soit Π, une projection orthogonale sur une face du prisme dans lequel un polyomino de volume minimal P est inscrit. Alors, $\Pi(P)$ correspond à P sans ses pas dans une certaine direction. Ainsi, les pas dans $\Pi(P)$ sont comme ceux décrits à la remarque 5.5.1, c'est-à-dire qu'il n'y en a pas deux pour une même contribution. De plus, $\Pi(P)$ est connexe sinon P n'est pas connexe. Donc, selon la remarque 5.5.1, $\Pi(P)$ est d'aire minimale.

■

Remarque 5.5.2 Selon le lemme 5.5.1, la projection orthogonale d'un polyomino de volume minimal sur une face de son prisme circonscrit a donc la forme *equerre* × *escalier* × *equerre*. △

Théorème 5.5.1 *La série génératrice des polyominos de volume minimal est*

$$P_{3Dmin}(x, y, z) = Diag(x, y, z) + P_{2DX2D}(x, y, z) + CG(x, y, z).$$

Preuve Pour chaque polyomino de volume minimal P, considérons sa projection orthogonale $\Pi(P)$ sur la face du haut du prisme dans lequel il est inscrit. Aussi, considérons ses cellules de contact et leur projection orthogonale $\Pi(c_1)$, $\Pi(c_2)$ sur cette même face. Nous pouvons alors partitionner l'ensemble des polyominos de volume minimal selon l'emplacement de $\Pi(c_1)$ et $\Pi(c_2)$ dans $\Pi(P)$.L'emplacement de ces cellules forme plusieurs classes dont les principales sont présentées à la figure 35.

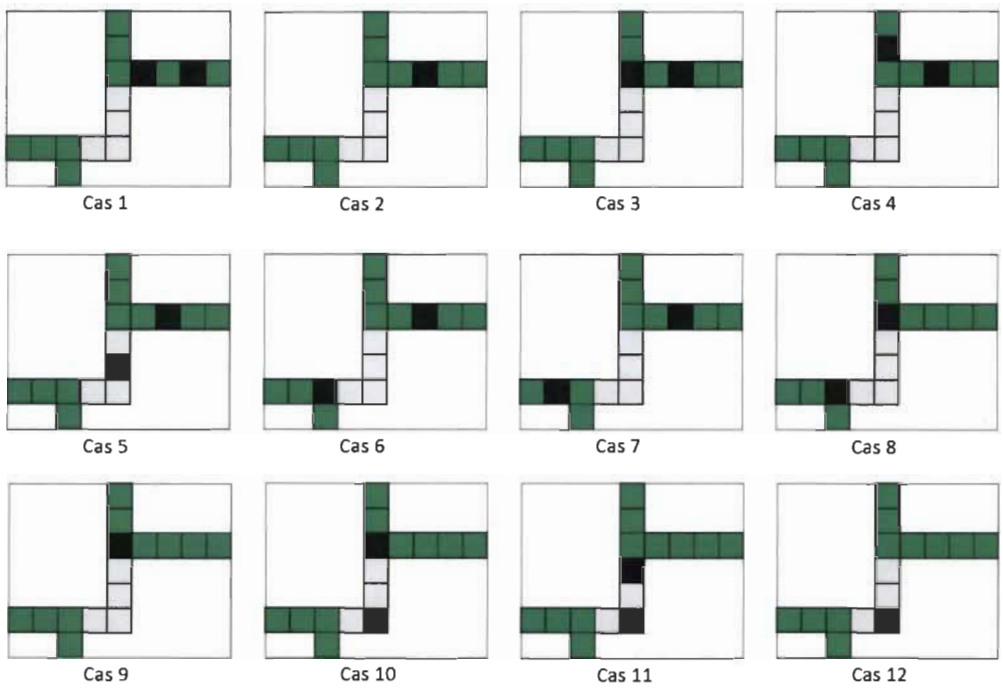

Figure 35 Positions possibles des cellules de contact

Nous prouverons ce théorème en montrant que pour chacun de ces 12 cas, nous pouvons construire les polyominos de cette classe et trouver que ceux-ci sont soit diagonaux, soit des 2DX2D, soit des croix gauches. Il existe d'autres cas, mais nous pouvons les inclure par symétrie comme un des 12 cas traités.

Dans chacun des cas, nous débuterons la construction des polyominos 3D par un escalier 3D qui relie les cellules de contact. Il n'est pas difficile de se convaincre que

les cellules de contact sont nécessairement reliées par un escalier. Pour un ensemble de cellules reliant l'escalier à une face du prisme autre que le plancher et le plafond, ces cellules sont dans un même plan parallèle au plafond.

Cas 1 : Un fois que les cellules de contact sont reliées par un escalier 3D, il faut ajouter des rangées de cellules partant de cet escalier pour compléter la partie dont la projection est le bras horizontal de l'équerre supérieure sur la figure 35 (cas 1). À cette étape de la construction, nous avons un polyomino d'aire minimale placé à la verticale et ce qui reste à construire est dans un plan orthogonal au plan contenant le début de la construction. Ainsi, les polyominos de cette classe sont des polyominos 2DX2D.

Cas 2 : Ce cas se traite exactement comme le cas 1.

Cas 3 : La partie de P dont la projection est le bras horizontal de l'équerre supérieure sur la figure 35 (cas 3) est un polyomino coin. De plus, la partie qui correspond au bras vertical de cette même équerre est un pilier. Ce pilier et ce polyomino coin forment une équerre 3D. Il y a un autre pilier qui relie la cellule de contact dont la projection est le coin de l'équerre supérieure. Ce pilier forme une autre équerre 3D avec le polyomino coin dont la projection est l'escalier (en gris) et l'équerre inférieure. Ainsi, les polyominos de cette classe sont diagonaux, car ils ont la forme *equerre* $\times$ *escalier* $\times$ *equerre*.

Cas 4 : On trouve un polyomino coin dont la projection est le bras horizontal de l'équerre supérieure, un autre polyomino coin dont la projection est le bras vertical de l'équerre supérieure et un dernier polyomino coin dont la projection est l'escalier (en gris) et l'équerre inférieure. Cela forme une croix gauche.

Cas 5 : Disons, sans perte de généralité, que la cellule de contact qui touche le bas du prisme est celle dont la projection est dans l'escalier (en gris). Alors la partie dont la projection est dans le sous rectangle dont le coin supérieur droit est la projection de cette cellule de contact consiste en une équerre 3D. De plus, la partie dont la projection

est le bras horizontal de l'équerre supérieure est un polyomino coin. La projection du coin de ce polyomino coin est le coin de l'équerre supérieure et le bras vertical de cette équerre est la projection d'un pilier. Ce pilier et ce dernier polyomino coin forment une équerre 3D. Nous retrouvons la forme $equerre3D \times escalier \times equerre3D$ des diagonaux.

Cas 6 : Ce cas ressemble beaucoup au cas 5. La différence est que l'équerre inférieure peut être la projection d'un tripode ou d'une équerre 3D. Les polyominos de cette classe sont diagonaux.

Cas 7 : Ce cas aussi ressemble au cas 5. Ici, l'équerre inférieure est la projection d'une équerre 3D. Les polyominos de cette classe sont des diagonaux.

Cas 8 : Ici, les équerres sont des projections de tripodes ou d'équerres 3D. Nous retrouvons la forme de certains polyominos diagonaux.

Cas 9 : Les polyominos de cette classe sont des diagonaux sur deux diagonales (voir la figure 31.).

Cas 10 : Ce cas se traite comme le cas 5. Les polyominos de cette classe sont diagonaux.

Cas 11 : Ce cas se traite comme le cas 5. Les polyominos de cette classe sont diagonaux.

Cas 12 : Ce cas se traite comme le cas 9. Les polyominos de cette classe sont diagonaux.

■

5.6 Des formules provenant de la série génératrice des polyominos de volume minimal

Si nous posons $x = y = z$ dans la série génératrice des polyominos de volume minimal, nous obtenons

$$P_{3D,min}(x) = \frac{x^3(72x^{10} + 36x^9 + 510x^8 - 1117x^7 + 1276x^6 - 1155x^5 + 710x^4 - 293x^3 + 81x^2 - 13x + 1)}{(1-3x)(1-2x)^3(1-x)^7}.$$

Les contributions en x, y ou z étant confondues, le coefficient devant x^{n+2} dans cette série est égal au nombre de polyomino de volume minimal à n cellules. Comme cette série n'a qu'une seule variable, il est facile d'obtenir une formule directe de celle-ci en utilisant un logiciel de calcul symbolique. Avec *Maple*, on peut utiliser le package *genfunc* pour faire cela. On trouve que le nombre de polyomino de volume minimal à n cellules est

$$\begin{aligned} p_{3D,min}(n) = {} & \frac{11^2 \cdot 3^{n+1}}{16} + 2^{n+2}(4n^2 - 125n + 741) \\ & + \frac{3n^6}{40} - \frac{9n^5}{10} - \frac{7n^4}{2} - \frac{133n^3}{2} - \frac{1931n^2}{5} - \frac{31727n}{20} - \frac{47739}{16}. \end{aligned}$$

Le tableau suivant donne le nombre de polyominos de volume minimal à n cellules.

n	1	2	3	4	5	6	7	8	9	10
$p_{3D,min}(n)$	1	3	15	83	450	2295	10834	47175	190407	719243

Tableau III Nombre de polyominos de volume minimal n

Aucune formule directe n'a été trouvée pour le nombre de polyominos de volume minimal inscrits dans un prisme cubique de dimensions $n \times n \times n$. Cependant, des échanges avec Guoce Xin [12] ont permis de trouver une récurrence donnant le nombre de croix gauches inscrites dans un prisme cubique de dimensions $n \times n \times n$. Dans la

récurrence qui suivra, ce nombre sera $G(n)$. Pour trouver cette récurrence, on doit trouver le terme constant de la série génératrice des croix gauches ayant subit le changement de variable $z = \frac{t}{xy}$. Une méthode faisant appel à l'usage de *Maple* et utilisée par Guoce Xin conduit à cette récurrence. Cette récurrence vient d'une équation différentielle linéaire aux coefficients polynomiaux. On passe de l'équation différentielle à la récurrence par la fonction $diffeqtorec$ du package $gfun$, mais cette équation différentielle est le résultat d'un long calcul sortant du cadre de ce mémoire. Cette récurrence est

$$
\begin{aligned}
0 = {} & \left(-63180 - 27270\,n - 3870\,n^2 - 180\,n^3\right) G(n+8) \\
& + \left(2947200 + 1484232\,n + 245775\,n^2 + 13287\,n^3\right) G(n+7) \\
& + \left(-45660336 - 25264262\,n - 4601580\,n^2 - 274894\,n^3\right) G(n+6) \\
& + \left(323508600 + 199647652\,n + 41274228\,n^2 + 2831360\,n^3\right) G(n+5) \\
& + \left(-1151571096 - 855054420\,n - 221109420\,n^2 - 19232064\,n^3\right) G(n+4) \\
& + \left(2171968224 + 2263835216\,n + 826292754\,n^2 + 99180346\,n^3\right) G(n+3) \\
& + \left(-2574543312 - 4129969996\,n - 2114426268\,n^2 - 343753544\,n^3\right) G(n+2) \\
& + \left(2212044480 + 4173204480\,n + 2698553280\,n^2 + 584148480\,n^3\right) G(n+1) \\
& + \left(-26657280 - 173272320\,n - 359873280\,n^2 - 239915520\,n^3\right) G(n) \\
& - 56215934976 - 28809784320\,n - 13121740800\,n^2 + 61314703104\,n^3 \\
& + 128260335168\,n^4 + 80378313984\,n^5 + 5311825344\,n^6
\end{aligned}
$$

où $G(3) = 64$, $G(4) = 4864$,$G(5) = 118784$,$G(6) = 1826816$,$G(7) = 22172928$, $G(8) = 234951168$,$G(9) = 2289445120$ et $G(10) = 21124466176$.

CONCLUSION

En définissant de nouvelles classes de polyominos, il est possible d'améliorer certains algorithmes énumérant les polyominos généraux. Toutefois, le problème d'énumération des polyominos généraux reste ouvert. On peut tenter d'énumérer les polyominos ayant i cellules de plus que les polyominos minimaux. Cela a été fait pour les polyominos ayant $b+k$ cellules dans un rectangle de dimensions $b \times k$. Voyez l'article *Enumeration of polyominos inscribed in a rectangle* de A.Goupil, H.Cloutier et F.Nouboud [7]. On peut aussi énumérer les polyominos ayant $bk - j$ cellules où j est le nombre de cases vides du rectangle $b \times k$ où ces polyominos sont inscrits. L'énumération des polyominos de ces nouvelles classes ne peut qu'améliorer l'efficacité d'un algorithme énumérant les polyominos généraux comme celui du chapitre 3.

Il y a une formule au chapitre 5 qui donne le nombre de polyominos de volume minimal ayant un nombre de cellules fixé, mais aucune formule directe donnant le nombre de polyominos de volume minimal inscrits dans un prisme de dimensions $b \times h \times k$ n'a été trouvée. La méthode utilisée par G. Xin [12] montre que dans certains cas, il existe un moyen d'obtenir une récurrence linéaire d'une série génératrice à plusieurs variables. Aussi, les polyominos de volume minimal ne peuvent être tous obtenus d'une généralisation des polyominos minimaux 2D. C'est-à-dire que les polyominos minimaux 3D ne sont pas tous des polyominos diagonaux.

ANNEXE 1

Programme énumérant les polyominos à forme fixée

Le programme présenté dans cette annexe permet d'énumérer les polyominos à forme fixée d'une aire donnée. Ce programme peut encore être amélioré de différentes manières, notamment en l'écrivant pour exécuter des tâches simultanément (en parallèle). Aussi, il est possible de garder en mémoire les différents résultats de la récurrence utilisée plutôt que de les recalculer. Cela n'est pas fait dans ce programme écrit à partir de ce qui est présenté au chapitre 3. Ce programme est écrit dans le langage C++. Les fichiers d'entête ont été volontairement omis, car ceux-ci peuvent être facilement reconstitués.

```
//*********************************************************************//
// Point d'entrée du programme: méthode main()                         //
//*********************************************************************//

#include "stdafx.h"
#include "stdio.h"
#include "ConversionEnColonne.h"
#include "LesComplementaires.h"
#include "Compteur.h"
#include "LesPrecolonnes.h"
#include "CompteurPourPrecolonne.h"
#include "math.h"
#include "ConstructionDesComplementaires.h"
#include "ConstructionDesPrecolonnes.h"
#include "LesColonnesADroite.h"
#include "ConstructionDesColonnesADroite.h"
#include "Somme.h"
#include "Recurrence.h"

int main()
{
    std::cout<<"Entrer l'aire des polyominos (Cela correspond aussi à
               la
               borne de hauteur.)\n \n";
```

```
    int aire;
    std::cin>> aire;

    int dimensionTableau;
    if(aire % 2==0)
    {
        dimensionTableau=(int)aire/2+1;
    }
    else
    {
        dimensionTableau=(int)(aire+1)/2+1;
    }

    int *colonneVide=new int[2*dimensionTableau*dimensionTableau];
    for(int i=0; i<2*dimensionTableau*dimensionTableau; i++)
    {
        *(colonneVide+i)=0;
    }

    int hauteur=aire;

    int NbrDePolyominos1=Recurrence::
                                calculNbrPolyominos(colonneVide,
                                                    aire,
                                                    dimensionTableau,
                                                    aire, hauteur);

    int NbrDePolyominos2=Recurrence::calculNbrPolyominos(colonneVide,
                                                    aire,
                                                    dimensionTableau,
                                                    aire-1, hauteur);

    std::cout<<"Il y a "<< NbrDePolyominos1 - NbrDePolyominos2
             <<" polyominos d'aire "<< aire<<". \n";


    delete[] colonneVide;

    char fin;
    std::cin>> fin;
    return 0;
}
```

```
//************************** Fin du fichier ************************//

//*********************************************************************//
// Classe: Compteur                                                    //
// Les objets de cette classe sont des compteurs utilisés pour énumé-//
// rer les colonnes complémentaires immédiatement à droite d'une cer-//
// taine colonne lors de la construction d'un polyomino.               //
//*********************************************************************//

#include <iostream>
#include "Compteur.h"
#include "math.h"

Compteur::Compteur(int *lesVides, int dimTableau, int bidon)
{
    int i=0; //pour initialiser un compteur
    while(*(lesVides+i*dimTableau*2)!=0)
    {
        i++;
    }

    int nbrDeVides=i;

    bases=new int[nbrDeVides];
    decimales=new int[nbrDeVides];

    //pour initialiser un compteur

    for(int j=0; j<nbrDeVides; j++)
    {
        *(bases+j)=(int)pow(2.0, *(lesVides+j*dimTableau*2));
    }

    for(int i=0; i<nbrDeVides; i++)
    {
        *(decimales+i)=0;
    }

    taille=nbrDeVides;
}

Compteur::~Compteur(void)
{
```

```
    delete[] decimales;
    delete[] bases;
}

int* Compteur::getBases()
{
    return bases;
}

int* Compteur::getDecimales()
{
    return decimales;
}

bool Compteur::estAuMax()
{
    for(int i=0; i<taille; i++)
    {
        if(*(decimales+i)!=*(bases+i)-1)
        {
            return false;
        }
    }
    return true;
}

void Compteur::incrementer()
{
    int position=trouveLentierAaugmenter();
    if(position!=-1)
    {
        *(decimales+position)+=1;
        for(int i=0; i<position; i++)
        {
            *(decimales+i)=0;
        }
    }
}

int Compteur::trouveLentierAaugmenter()
{
    for(int i=0; i<taille; i++)
{
```

```
        if(*(decimales+i)!=*(bases+i)-1)
        {
            return i; //i est la position de l'entier à augmenter
        }
    }
    return -1;
}

//************************* Fin du fichier ************************//

//*********************************************************************//
// Classe: CompteurPourPrecolonne                                      //
// Les objets de cette classe sont des compteurs utilisés pour énumé-//
// rer les précolonnes immédiatement à droite d'une certaine colonne //
// lors de la construction d'un polyomino.                             //
//*********************************************************************//

#include <iostream>
#include "CompteurPourPrecolonne.h"
#include "math.h"

CompteurPourPrecolonne::CompteurPourPrecolonne(int *lesBases,
                                               int nbrDeCases)
{
    bases=new int[nbrDeCases];
    decimales=new int[nbrDeCases];
    taille=nbrDeCases;
    for(int i=0; i<nbrDeCases; i++)
    {
        *(bases+i)=(int)pow((double)2,(double)*(lesBases+i));
        *(decimales+i)=1;
    }

}

CompteurPourPrecolonne::~CompteurPourPrecolonne(void)
{
    delete[] bases;
    delete[] decimales;
}

bool CompteurPourPrecolonne::estAuMax()
{
```

```
    for(int i=0; i<taille; i++)
    {
        if(*(decimales+i)!=*(bases+i)-1)
        {
            return false;
        }
    }
    return true;
}

void CompteurPourPrecolonne::incrementer()
{
    int position=trouveLentierAaugmenter();
    if(position!=-1)
    {
        *(decimales+position)+=1;
        for(int i=0; i<position; i++)
        {
            *(decimales+i)=1;
        }
    }
}

int CompteurPourPrecolonne::trouveLentierAaugmenter()
{
    for(int i=0; i<taille; i++)
    {
        if(*(decimales+i)!=*(bases+i)-1)
        {
            return i; //i est la position de l'entier à augmenter
        }
    }
    return -1;
}

//************************* Fin du fichier ************************//


//*****************************************************************//
// Classe: ConstructionDesColonnesADroite                          //
// Cette classe fait l'ensemble des colonnes immédiatement à droite //
// d'une colonne d'un polyomino en construction.                   //
//*****************************************************************//
```

```
#include <iostream>
#include "ConstructionDesColonnesADroite.h"
#include "ConstructionDesPrecolonnes.h"
#include "ConstructionDesComplementaires.h"
#include "Somme.h"
#include "ConversionEnColonne.h"

ConstructionDesColonnesADroite::ConstructionDesColonnesADroite(void)
{
}

ConstructionDesColonnesADroite::~ConstructionDesColonnesADroite(void)
{
}

int **ConstructionDesColonnesADroite::
                            faireLensemble(int ***lesColComp,
                                           int **ensembleDePrecolonne,
                                           int *colonne,
                                           int dimTableau, int aire)
{
    int nbrPreColonnes=ConstructionDesPrecolonnes::
                               getNbrPreColonnes(colonne, dimTableau);

    int nbrComplementaires=ConstructionDesComplementaires::
                                    getNbrEnsDEnsembleDeComp(colonne,
                                                             dimTableau,
                                                             aire);

    int nbrDeVidesElementaires=ConstructionDesComplementaires::
                                     getNbrDeVides(colonne, dimTableau,
                                                   aire);

    int  **lEnsembleDesColonnesADroite=
                           new int*[nbrPreColonnes*nbrComplementaires];

    for(int i=0; i<nbrPreColonnes; i++)
    {
        for(int j=0; j<nbrComplementaires; j++)
        {
            int *precolonne;
            precolonne=new int[2*dimTableau*dimTableau];
```

```
            for(int k=0; k<2*dimTableau*dimTableau; k++)
            {
                *(precolonne+k)=*(*(ensembleDePrecolonne+i)+k);
            }

            *(lEnsembleDesColonnesADroite+i*nbrComplementaires+j)=
                    Somme::add(precolonne, *(lesColComp+j), dimTableau,
                               nbrDeVidesElementaires, 0);

            delete[] precolonne;
        }
    }
    return lEnsembleDesColonnesADroite;
}

//************************* Fin du fichier ************************//

//*********************************************************************//
// Classe: ConstructionDesComplementaires                              //
// Cette classe fait l'ensemble des colonnes complementaires immédia-//
// tement à droite d'une colonne d'un polyomino en construction. Cet-//
// te classe contient aussi une méthode qui donne le nombre de manië-//
// res de construire ces colonnes complémentaires. Une autre méthode //
// compte les vides d'une colonne donnée.                              //
//*********************************************************************//

#include <iostream>
#include "ConstructionDesComplementaires.h"
#include "LesComplementaires.h"
#include "ConversionEnColonne.h"
#include "math.h"

ConstructionDesComplementaires::ConstructionDesComplementaires(void)
{
}

ConstructionDesComplementaires::~ConstructionDesComplementaires(void)
{
}

int ***ConstructionDesComplementaires::
                            faireComplementaires(int *colonne,
```

```
                                                    int dimensionTableau,
                                                    int aire)
{
    //Conversion en entier
    int entierDeColonne=LesComplementaires::
                                    convertisEnEntier(colonne,
                                                      dimensionTableau);

    //Conversion en binaire complémentaire
    int *colonneBinaire=
    LesComplementaires::
              convertisEnBinaireComplementaire(entierDeColonne, aire);

    //Conversion en complémentaire sous forme matricielle
    int *colonneComplementaire=
        ConversionEnColonne::enColonneComplementaire(colonneBinaire,
                                                      dimensionTableau,
                                                      aire);

    delete[] colonneBinaire;

    Compteur c=Compteur(colonneComplementaire, dimensionTableau, 0);

    int nbrDensemblesDeColComp=1;
    for(int i=0; i<c.taille; i++)
    {
        nbrDensemblesDeColComp*= *(c.bases+i);
    }

    //Construction de l'ensemble d'ensembles de colonnes
    //complementaires

    int ***lesColonnesComplementaires=
        new int **[nbrDensemblesDeColComp];

    int j=0;
    while( !(c.estAuMax()) )
    {
        *(lesColonnesComplementaires+j)=
                  LesComplementaires::faireComp(c.taille,
                                                c.decimales,
                                                colonneComplementaire,
                                                dimensionTableau,
```

```
                                                    aire);
        j++;
        c.incrementer();
    }

//Construction d'un ensemble de colonnes complémentaires avec c max.

    *(lesColonnesComplementaires+j)=
                    LesComplementaires::faireComp(c.taille,
                                                    c.decimales,
                                                    colonneComplementaire,
                                                    dimensionTableau,
                                                    aire);

    delete[] colonneComplementaire;

    return lesColonnesComplementaires;
}

int ConstructionDesComplementaires::
            getNbrEnsDEnsembleDeComp(int *colonne, int dimensionTableau,
                                     int aire)
{
    //Conversion en entier
    int entierDeColonne=LesComplementaires::
                                    convertisEnEntier(colonne,
                                                      dimensionTableau);

    //Conversion en binaire complémentaire
    int *colonneBinaire=
        LesComplementaires::
                convertisEnBinaireComplementaire(entierDeColonne, aire);

    int i=0;
    int nbrDeCellulesVides=0;
    while(i<aire)
    {
        if(*(colonneBinaire+i)==1)
        {
            nbrDeCellulesVides++;
        }
        i++;
    }
```

```
    delete[] colonneBinaire;

    return (int)pow((double)2, (double)nbrDeCellulesVides);
}

int ConstructionDesComplementaires::getNbrDeVides(int *colonne,
                                                   int dimTableau,
                                                   int aire)
{
    //Conversion en entier
    int entierDeColonne=LesComplementaires::
                              convertisEnEntier(colonne, dimTableau);

    //Conversion en binaire complémentaire
    int *colonneBinaire=
        LesComplementaires::
               convertisEnBinaireComplementaire(entierDeColonne, aire);

    int nbrDeVide=0;
    int i=0;
    while(i<aire)
    {
        if(*(colonneBinaire+i)==1)
        {
            nbrDeVide++;
            while(*(colonneBinaire+i)==1)
            {
                i++;
            }
        }
        i++;
    }

    delete[] colonneBinaire;

    return nbrDeVide;
}

//************************* Fin du fichier ************************//

//*****************************************************************//
// Classe: ConstructionDesPrecolonnes                             //
```

```
// Cette classe fait l'ensemble des précolonnes immédiatement à droi-//
// te d'une colonne d'un polyomino en construction. Cette classe con-//
// tient aussi une méthode permettant d'énumérer ces précolonnes.    //
//*****************************************************************//

#include <iostream>
#include "ConstructionDesPrecolonnes.h"
#include "LesPrecolonnes.h"
#include "math.h"
#include "ConversionEnColonne.h"

ConstructionDesPrecolonnes::ConstructionDesPrecolonnes(void)
{
}

ConstructionDesPrecolonnes::~ConstructionDesPrecolonnes(void)
{
}

int **ConstructionDesPrecolonnes::
               faireLesPrecolonnes(int *colonne, int dimensionTableau,
                                   int aire)
{
    int *tableauDesBasesPourCompteur=
              ConstructionDesPrecolonnes::
                  fabriquerTableauDesBases(colonne, dimensionTableau);

    int nbrDeColonnesElem=ConstructionDesPrecolonnes::
                            getNbrColElem(colonne, dimensionTableau);

    CompteurPourPrecolonne compteur(tableauDesBasesPourCompteur,
                                    nbrDeColonnesElem);

    int nbrDePreColonnes=1;
    for(int i=0; i<nbrDeColonnesElem; i++)
    {
        nbrDePreColonnes *= (*(compteur.bases+i)-1);
    }

    delete[] tableauDesBasesPourCompteur;

    int **lEnsDesPrecolonnes=new int*[nbrDePreColonnes];
```

```
int noDePrecolonne=0;

while(!(compteur.estAuMax()))
{
    int **precolonnesDesordonnees;
    precolonnesDesordonnees=new int*[nbrDeColonnesElem];

    for(int i=0; i<nbrDeColonnesElem; i++)
    {
        *(precolonnesDesordonnees+i)=
            LesPrecolonnes::
                 versPrecolonneElementaire(*(compteur.decimales+i),
                                           colonne, i, aire,
                                           dimensionTableau);
    }

    int *unePrecolonne =
           LesPrecolonnes::fairePrecolonne(precolonnesDesordonnees,
                                           dimensionTableau,
                                           nbrDeColonnesElem);

    (*(lEnsDesPrecolonnes+noDePrecolonne))=
        new int[2*dimensionTableau*dimensionTableau];

    for(int i=0; i<2*dimensionTableau*dimensionTableau; i++)
    {
        *(*(lEnsDesPrecolonnes+noDePrecolonne)+i)=
                                                *(unePrecolonne+i);
    }

    noDePrecolonne++;
    compteur.incrementer();

    delete[] unePrecolonne;

    for(int i=0; i<nbrDeColonnesElem; i++)
    {
        delete[] *(precolonnesDesordonnees+i);
    }
    delete[] precolonnesDesordonnees;
}

//construction de la colonne pour compteur au max.
```

```
    int **precolonnesDesord;
    precolonnesDesord=new int*[nbrDeColonnesElem];

    for(int i=0; i<nbrDeColonnesElem; i++)
    {
        *(precolonnesDesord+i)=
            LesPrecolonnes::
                    versPrecolonneElementaire(*(compteur.decimales+i),
                                              colonne, i, aire,
                                              dimensionTableau);
    }

    int *unePrecol=LesPrecolonnes::fairePrecolonne(precolonnesDesord,
                                                   dimensionTableau,
                                                   nbrDeColonnesElem);

    (*(lEnsDesPrecolonnes+noDePrecolonne))=
        new int[2*dimensionTableau*dimensionTableau];

    for(int i=0; i<2*dimensionTableau*dimensionTableau; i++)
    {
        *(*(lEnsDesPrecolonnes+noDePrecolonne)+i)=*(unePrecol+i);
    }

    delete[] unePrecol;

    for(int i=0; i<nbrDeColonnesElem; i++)
    {
        delete[] *(precolonnesDesord+i);
    }
    delete[] precolonnesDesord;

    return lEnsDesPrecolonnes;
}

int *ConstructionDesPrecolonnes::
                fabriquerTableauDesBases(int *colonne, int dimTableau)
{
    int nbrDeColonnesElem=ConstructionDesPrecolonnes::
                                    getNbrColElem(colonne, dimTableau);

    int *basesCompteur;
```

```
    if(nbrDeColonnesElem <= 0)
    {
        basesCompteur = new int[2];
    }
    else
    {
        basesCompteur=new int[nbrDeColonnesElem];

        for(int j=0; j<nbrDeColonnesElem; j++)
        {
            int baseJ=0;
            for(int k=0; *(colonne+j*2*dimTableau+2*k)!=0; k++)
            {
                baseJ += *(colonne+j*2*dimTableau+2*k);
            }
            *(basesCompteur+j)=baseJ;

        }
    }

    return basesCompteur;
}

int ConstructionDesPrecolonnes::getNbrColElem(int *colonne,
                                              int dimTableau)
{
    int i=0;
    while(*(colonne+2*dimTableau*i)!=0 && i<dimTableau)
    {
        i++;
    }
    return i;
}

int ConstructionDesPrecolonnes::getNbrPreColonnes(int *colonne,
                                                  int dimTableau)
{
    int *tableauDesBases=fabriquerTableauDesBases(colonne, dimTableau);

    int tailleTableauDesBases=getNbrColElem(colonne, dimTableau);

    if(tailleTableauDesBases == 0)
    {
```

```
        delete[] tableauDesBases;
        return 1;
    }
    int nbrDePreColonnes=1;
    for(int i=0; i<tailleTableauDesBases; i++)
    {
        nbrDePreColonnes *= (int)(pow((double)2,
                                      (double) *(tableauDesBases+i))
                                  -1);
    }

    delete[] tableauDesBases;

    if(nbrDePreColonnes == 0) std::cout<<"ERREUR";

    return nbrDePreColonnes;
}

//************************** Fin du fichier ************************//

//*********************************************************************//
// Classe: ConversionEnColonne                                         //
// Cette classe contient une  méthode qui donne le complément d'une  //
// colonne. Elle contient aussi une méthode qui transforme une colon-//
// ne en une chaîne binaire et une autre méthode qui affiche l'ensem-//
// ble d'ensemble de couple décrivant une colonne donnée.             //
//*********************************************************************//

#include <iostream>
#include "ConversionEnColonne.h"
#include "math.h"
#include "stdio.h"


ConversionEnColonne::ConversionEnColonne(void)
{
}

ConversionEnColonne::~ConversionEnColonne(void)
{
}

int *ConversionEnColonne::
```

```
                convertisEnComplementaire(int n, int dimensionTableau,
                                          int borne)
{
    int *entierBinaire=enBinaire(n, borne);
    int *colonneComplementaire=
                            enColonneComplementaire(entierBinaire,
                                                    dimensionTableau,
                                                    borne);

    delete[] entierBinaire;

    return colonneComplementaire;
}

int *ConversionEnColonne::enBinaire(int n, int borne)
{
    int *formeBinaire;
    formeBinaire=new int[borne];

    int i=0;
    int m=n;
    while(m > 0)
    {
        *(formeBinaire+i)= m % 2;
        m-=*(formeBinaire+i);
        m/=2;
        i++;
    }

    while(i<borne)
    {
        *(formeBinaire+i)=0;
        i++;
    }

    return formeBinaire;
}

int *ConversionEnColonne::enColonneComplementaire(int *entierBinaire,
                                                  int dimensionTableau,
                                                  int borne)
{
    int *colonne;
```

```
    colonne=new int[2*dimensionTableau*dimensionTableau];

    for(int i=0; i<2*dimensionTableau*dimensionTableau; i++)
    {
        *(colonne+i)=0;
    }

    int CES=0;
    int position=0;

    while((*(entierBinaire+position)==1 ||
          *(entierBinaire+position)==0)
          && position<borne)
    {
        if(*(entierBinaire+position)==1)
        {
            int positionCES=position+1;
            int hauteurCES=0;

            while(*(entierBinaire+position)==1 && position<borne)
            {
                hauteurCES ++;
                position ++;
            }

            *(colonne+CES*dimensionTableau*2)=hauteurCES;
            *(colonne+CES*dimensionTableau*2+1)=positionCES;
            CES ++;
        }

        position ++;
    }

    return colonne;
}

void ConversionEnColonne::afficher(int *colonne, int dimensionTableau)
{
    std::cout<<"{";

    for(int i=0;
        i<dimensionTableau &&*(colonne+i*dimensionTableau*2)!=0;
        i++)
```

```
    {
        std::cout<<"{";

        int j=0;
        while(*(colonne+i*dimensionTableau*2+j*2)!=0 &&
                                                    j<dimensionTableau)
        {
            std::cout<<"("<<*(colonne+i*dimensionTableau*2+j*2)<<","
                     <<*(colonne+i*dimensionTableau*2+j*2+1)<<")";
            j++;
        }
        std::cout<<"}";
    }
    std::cout<<"}";
}

//************************* Fin du fichier ************************//

//*********************************************************************//
// Classe: LesColonnesADroite                                          //
// Cette classe ne contient qu'une méthode. Cette méthode donne le     //
// nombre de cellules d'une colonne donnée.                            //
//*********************************************************************//

#include <iostream>
#include "LesColonnesADroite.h"
#include "ConversionEnColonne.h"

LesColonnesADroite::LesColonnesADroite(void)
{
}
LesColonnesADroite::~LesColonnesADroite(void)
{
}

int LesColonnesADroite::getNbrCel(int *tableauDeLaColonne, int dimTab)
{
    int resultat=0;
    for(int i=0; i<2*dimTab*dimTab; i+=2)
    {
        resultat += *(tableauDeLaColonne+i);
    }
```

```
    return resultat;
}

//************************* Fin du fichier ************************//

//*********************************************************************//
// Classe: LesComplementaires                                          //
// Cette classe permet principalement de fabriquer un ensemble déter-//
// miné de colonnes complémentaires associé à un ensemble de vides   //
// élémentaires.                                                       //
//*********************************************************************//

#include <iostream>
#include "LesComplementaires.h"
#include "ConversionEnColonne.h"
#include "math.h"
#include "Compteur.h"

LesComplementaires::LesComplementaires()
{
}

LesComplementaires::~LesComplementaires(void)
{
}

int **LesComplementaires::faireComp(int taille, int *decimales,
                                    int *colonne, int dimTab, int aire)
{
    int **ensembleDeColonnesComplementaires;
    if(taille != 0)
    {
        ensembleDeColonnesComplementaires=new int*[taille];

        int entierDeColonneComplementaire=0;
        for(int i=0; i<taille; i++)
        {
            entierDeColonneComplementaire=
                    *(decimales+i)*(int)pow(2.0,
                                        *(colonne+i*dimTab*2+1)-1);
                       //Translation de la i-ème composante du compteur

            *(ensembleDeColonnesComplementaires+i)=
```

```
                ConversionEnColonne::
                    convertisEnComplementaire(
                                        entierDeColonneComplementaire,
                                        dimTab,
                                        aire);
        }
    }

    if(taille==0)
    {
        ensembleDeColonnesComplementaires=new int*[1];
        *(ensembleDeColonnesComplementaires)=new int[2*dimTab*dimTab];

        for(int i=0; i<2*dimTab*dimTab; i++)
        {
            *(*(ensembleDeColonnesComplementaires)+ i)=0;
        }
    }

return ensembleDeColonnesComplementaires;
}

int LesComplementaires::convertisEnEntier(int *colonne, int dimTableau)
{
    int entierAretourner=0;
    for(int i=0; i<dimTableau; i++)
    {
        for(int j=0;
            j<dimTableau*dimTableau &&
                                    *(colonne+i*dimTableau*2+j*2)!=0;
            j++)
        {
            entierAretourner +=
                        entierDeCES(*(colonne+i*dimTableau*2+j*2),
                                    *(colonne+i*dimTableau*2+j*2+1));
        }
    }

    return entierAretourner;
}

int LesComplementaires::entierDeCES(int hauteur, int position)
{
```

```
    int entierCES=0;
    for(int i=position; i<position+hauteur; i++)
    {
        entierCES += (int)pow((double)2,(double)i-1);
    }

    return entierCES;
}

int *LesComplementaires::
                convertisEnBinaireComplementaire(int entierAconvertir,
                                                 int borne)
{
    int *entierBinaire=ConversionEnColonne::
                                    enBinaire(entierAconvertir, borne);

    int i=0;
    while(i<borne)
    {
        if(*(entierBinaire+i)==1)
        {
            *(entierBinaire+i)=0;
        }
        else
        {
            *(entierBinaire+i)=1;
        }

        i++;
    }

    return entierBinaire;
}

//************************** Fin du fichier ************************//

//*****************************************************************//
// Classe: LesPrecolonnes                                          //
// Cette classe permet principalement de fabriquer une précolonne à //
// partir de précolonnes élémentaires désordonnées.                //
//*****************************************************************//

#include <iostream>
```

```
#include "LesPrecolonnes.h"
#include "math.h"
#include "ConversionEnColonne.h"
#include "CompteurPourPrecolonne.h"

LesPrecolonnes::LesPrecolonnes()
{
}
LesPrecolonnes::~LesPrecolonnes(void)
{
}

int *LesPrecolonnes::fairePrecolonne(int **precolonneDesordonnee,
                                     int dimTableau, int nbrDeColElem)
{
    int *tableauPrecolonne=new int[2*dimTableau*dimTableau];
    for(int i=0; i<2*dimTableau*dimTableau; i++)
    {
        *(tableauPrecolonne+i)=0;
    }

    int numeroColonne=0;
    int positionColonneActuelle=0;
    while(numeroColonne<nbrDeColElem)
    {
        int *precolonneElemSuivante=
                        getPrecolSuivante(positionColonneActuelle,
                                          precolonneDesordonnee,
                                          nbrDeColElem, dimTableau);

        for(int j=0; j<2*dimTableau; j++)
        {
            *(tableauPrecolonne+numeroColonne*2*dimTableau+j)=
                                        *(precolonneElemSuivante+j);
        }

        positionColonneActuelle=*(precolonneElemSuivante+1);
        numeroColonne++;

        delete[] precolonneElemSuivante;
    }

    return tableauPrecolonne;
```

```
}

int *LesPrecolonnes::versPrecolonneElementaire(int N, int *colonne,
                                               int colElem, int borne,
                                               int dimTableau)
{
    int resultat=0;

    for(int i=0; i<taille(colonne, colElem, dimTableau); i++)
    {
        int exposant=0;
        for(int j=0; j<i; j++)
        {
            exposant += *(colonne+colElem*dimTableau*2+2*j);
        }

        resultat+=(int)(((int)(floor(N/pow((double)2,
                                           (double)exposant)
                                     )
                               )
                %
                (int)pow((double)2,
                        (double) *(colonne+colElem*dimTableau*2+2*i))
                       )
                *pow((double)2,
                     (double)*(colonne+colElem*dimTableau*2+2*i+1)-1)
                       );
    }

    int *colonneBinaire=ConversionEnColonne::
                                           enBinaire(resultat, borne);

    int *colonneElementaireResultat;
    colonneElementaireResultat=new int[dimTableau*2];
    for(int i=0; i<dimTableau*2; i++)
    {
        *(colonneElementaireResultat+i)=0;
    }

    int CES=0;
    int position=0;

    while((*(colonneBinaire+position)==1 ||
```

```
            *(colonneBinaire+position)==0)
            && position<borne)
    {
        if(*(colonneBinaire+position)==1)
        {
            int positionCES=position+1;
            int hauteurCES=0;

            while(*(colonneBinaire+position)==1 && position<borne)
            {
                hauteurCES++;
                position++;
            }

            *(colonneElementaireResultat+2*CES)=hauteurCES;
            *(colonneElementaireResultat+2*CES+1)=positionCES;

            CES++;
        }

        position++;
    }

    delete[] colonneBinaire;

    return colonneElementaireResultat;
}

int LesPrecolonnes::taille(int *colonne, int colElem, int dimTableau)
{
    int i=0;
    while(*(colonne+colElem*dimTableau*2+i)!=0)
    {
        i++;
    }

    return (int)(i/2);
}

int LesPrecolonnes::nbrDeColonnesElementaires(int *colonne,
                                              int dimTableau)
{
    int i=0;
```

```
    while(*(colonne+i*2*dimTableau)!=0)
    {
        i++;
    }

    return i;
}

int *LesPrecolonnes::getPrecolSuivante(int positionColonneActuelle,
                                       int **precolonneDesordonnee,
                                       int nbrDeColElem, int dimTab)
{
    int *precolSuivante=new int[2*dimTab];
    int *tabDesPositions= new int[nbrDeColElem];

    for(int i=0; i<nbrDeColElem; i++)
    {
        *(tabDesPositions + i) = *(*(precolonneDesordonnee + i) + 1);
    }

    int ecartMin = 2*dimTab*dimTab;
    int nouvellePosition=-1;
    for(int i=0; i<nbrDeColElem; i++)
    {
        int ecartI = *(tabDesPositions + i) - positionColonneActuelle;
        if(ecartI > 0 && ecartI < ecartMin)
        {
            ecartMin = ecartI;
            nouvellePosition=i;
        }
    }

    for(int i=0; i<2*dimTab; i++)
    {
        *(precolSuivante + i) =
                    *(*(precolonneDesordonnee + nouvellePosition) + i);
    }

    delete[] tabDesPositions;

    return precolSuivante;
}
```

```
//************************** Fin du fichier ************************//

//*****************************************************************//
// Classe: Recurrence                                              //
// Cette classe applique principalement la récurrence permettant de //
// construire les polyominos colonne par colonne.                  //
//*****************************************************************//

#include <iostream>
#include "Recurrence.h"
#include "LesColonnesADroite.h"
#include "ConstructionDesPrecolonnes.h"
#include "ConstructionDesComplementaires.h"
#include "ConstructionDesColonnesADroite.h"
#include "ConversionEnColonne.h"


Recurrence::Recurrence(void)
{
}

Recurrence::~Recurrence(void)
{
}

int Recurrence::calculNbrPolyominos(int *premiereColonne,
                                    int nbrDeCellulesRestantes,
                                    int dimTab, int aire, int hauteur)
{
    int nbrColADroite=getNbrColADroite(premiereColonne, dimTab, aire);

    if(nbrDeCellulesRestantes > aire + 1)
    {
        std::cout<<"ERREUR NBR CEL";
        return 0;
    }

    if(nbrDeCellulesRestantes > 0)
    {
        int **lEnsembleDesColonnesSuivantes=
                                    faireEnsDeColADroite(premiereColonne,
                                                         dimTab, aire);
```

```
    int resultat=0;

    for(int i=0; i<nbrColADroite; i++)
    {
        if(LesColonnesADroite::
           getNbrCel(*(lEnsembleDesColonnesSuivantes +i), dimTab)!=0)
        {
            resultat +=
                calculNbrPolyominos(
                    *(lEnsembleDesColonnesSuivantes +i),
                    nbrDeCellulesRestantes -
                    LesColonnesADroite::
                         getNbrCel(*(lEnsembleDesColonnesSuivantes +i),
                                   dimTab),
                                   dimTab, aire, hauteur);
        }

        delete[] *(lEnsembleDesColonnesSuivantes + i);
    }

    delete[] lEnsembleDesColonnesSuivantes;

    return resultat;
    }

    if(nbrDeCellulesRestantes==0)
    {
        if(*(premiereColonne+2*dimTab)==0)
        {
            return 1;
        }

        return 0;
    }

    return 0;
}

int **Recurrence::faireEnsDeColADroite(int *premiereCol,
                                       int dimTab, int hauteur)
{
    int ***lesColonnesComp=
        ConstructionDesComplementaires::
```

```
                faireComplementaires(premiereCol, dimTab, hauteur);

int nbrComplementaires=ConstructionDesComplementaires::
                              getNbrEnsDEnsembleDeComp(premiereCol,
                                                       dimTab,
                                                       hauteur);

int **lEnsembleDesColonnesADroite;

if(*premiereCol == 0)
{
    int **lEnsembleDesPreC = new int*[1];
    *(lEnsembleDesPreC) =new int[2*dimTab*dimTab];
    for(int i=0; i< 2*dimTab*dimTab; i++)
    {
        *(*(lEnsembleDesPreC)+ i)=0;
    }

    lEnsembleDesColonnesADroite =
        ConstructionDesColonnesADroite::
                faireLensemble(lesColonnesComp, lEnsembleDesPreC,
                               premiereCol, dimTab, hauteur);

    delete[] *lEnsembleDesPreC;
    delete[] lEnsembleDesPreC;
}

if(*premiereCol != 0)
{
    int *tableauDesBasesPourCompteur=
        ConstructionDesPrecolonnes::
                    fabriquerTableauDesBases(premiereCol, dimTab);
    int nbrDeColonnesElem=ConstructionDesPrecolonnes::
                               getNbrColElem(premiereCol, dimTab);

    CompteurPourPrecolonne compteur(tableauDesBasesPourCompteur,
                                    nbrDeColonnesElem);

    int nbrDePreColonnes=1;
    for(int i=0; i<nbrDeColonnesElem; i++)
    {
        nbrDePreColonnes *= (*(compteur.bases+i)-1);
    }
```

```
    delete[] tableauDesBasesPourCompteur;

    int **lEnsembleDesPrecolonnes=
        ConstructionDesPrecolonnes::
                                    faireLesPrecolonnes(premiereCol,
                                                        dimTab,
                                                        hauteur);

    lEnsembleDesColonnesADroite =
        ConstructionDesColonnesADroite::
            faireLensemble(lesColonnesComp,
                           lEnsembleDesPrecolonnes,
                           premiereCol, dimTab, hauteur);

    int nbrPreColonnes=ConstructionDesPrecolonnes::
                            getNbrPreColonnes(premiereCol, dimTab);

    for(int i=0; i< nbrPreColonnes; i++)
    {
        delete[] *(lEnsembleDesPrecolonnes + i);
    }

    delete[] lEnsembleDesPrecolonnes;
}

int nbrDeVidesElementaires=
    ConstructionDesComplementaires::getNbrDeVides(premiereCol,
                                                  dimTab, hauteur);

if(nbrDeVidesElementaires == 0)
{
    nbrDeVidesElementaires = 1;
}

for(int i=0; i<nbrComplementaires; i++)
{
    for(int j=0; j<nbrDeVidesElementaires; j++)
    {
        delete[] *(*(lesColonnesComp + i) + j);
    }

    delete[] *(lesColonnesComp + i);
```

```
    }

    delete[] lesColonnesComp;

    return lEnsembleDesColonnesADroite;
}



int Recurrence::getNbrColADroite(int *premiereColonne, int dimTab,
                                 int hauteur)
{
    int nbrPreColonnes=ConstructionDesPrecolonnes::
                          getNbrPreColonnes(premiereColonne, dimTab);

    int nbrComplementaires=
         ConstructionDesComplementaires::
                            getNbrEnsDEnsembleDeComp(premiereColonne,
                                                     dimTab,
                                                     hauteur);

    return nbrPreColonnes*nbrComplementaires;
}

//************************** Fin du fichier ************************//

//*****************************************************************//
// Classe: Somme                                                   //
// Cette classe permet d'additionner un ensemble de colonnes complé- //
// mentaires à une précolonne.                                     //
//*****************************************************************//

#include <iostream>
#include "Somme.h"
#include "ConversionEnColonne.h"

Somme::Somme(void)
{
}

Somme::~Somme(void)
{
}
```

```
int* Somme::add(int* precolonne, int **lEnsembleDeComp,
                int dimTableau, int nbrDeVide, int numeroDuVideActuel)
{
    if(nbrDeVide == 0)
    {
        int *laColonneEnConstruction=new int[2*dimTableau*dimTableau];
        for(int i=0; i<2*dimTableau*dimTableau; i++)
        {
            *(laColonneEnConstruction + i) = *(precolonne + i);
        }

        return laColonneEnConstruction;
    }

    int *resultat = new int[2*dimTableau*dimTableau];

    for(int i=0; i<2*dimTableau*dimTableau; i++)
    {
        *(resultat + i) = *(precolonne +i);
    }

    for(int i=0; i<nbrDeVide; i++)
    {
        int *resultatTemp;
        resultatTemp = addition(resultat, *(lEnsembleDeComp + i),
                                dimTableau);

        for(int j=0; j<2*dimTableau*dimTableau; j++)
        {
            *(resultat +j)=*(resultatTemp+j);
        }

        delete[] resultatTemp;
    }

    return resultat;
}

int* Somme::addition(int *laColonneEnConstruction,
                     int *laComplementaire, int dimTab)
{
    int *copieDeLaComplementaire = new int[2*dimTab*dimTab];
```

```
for(int in=0; in<2*dimTab*dimTab; in++)
    *(copieDeLaComplementaire + in) = *(laComplementaire + in);

int *retour1 = new int[2*dimTab*dimTab];
for(int i=0; i<2*dimTab*dimTab; i++)
{
    *(retour1 + i)=*(laColonneEnConstruction + i);
}

if(*copieDeLaComplementaire == 0)
{
    delete[] copieDeLaComplementaire;
    return retour1;
}

delete[] retour1;

bool contactInf=testContactInf(laColonneEnConstruction,
                               copieDeLaComplementaire, dimTab);

bool contactSup=testContactSup(laColonneEnConstruction,
                               copieDeLaComplementaire, dimTab);

int *retour2;
if(!contactInf && !contactSup)
{
    retour2 = additionSansContact(laColonneEnConstruction,
                                  copieDeLaComplementaire, dimTab);

    delete[] copieDeLaComplementaire;

    return retour2;
}

if(contactInf && !contactSup)
{
    retour2 = additionContactInf(laColonneEnConstruction,
                                 copieDeLaComplementaire, dimTab);

    delete[] copieDeLaComplementaire;

    return retour2;
```

```
    }

    if(!contactInf && contactSup)
    {
        retour2 = additionContactSup(laColonneEnConstruction,
                                     copieDeLaComplementaire, dimTab);

        delete[] copieDeLaComplementaire;

        return retour2;
    }

    if(contactInf && contactSup)
    {
        if(*(copieDeLaComplementaire+2*dimTab)==0)
        {
            retour2 = additionFusion(laColonneEnConstruction,
                                     copieDeLaComplementaire, dimTab);

            delete[] copieDeLaComplementaire;

            return retour2;
        }

        retour2 = additionContactSupInf(laColonneEnConstruction,
                                        copieDeLaComplementaire,
                                        dimTab);
        delete[] copieDeLaComplementaire;

        return retour2;
    }

    delete[] retour2;

    return laColonneEnConstruction;
}

bool Somme::testContactInf(int *laColonneEnConstruction,
                           int *laComplementaire, int dimTab)
{
    bool contact=false;

    int i=0;
```

```
    while(!(*(laColonneEnConstruction+i)==0 &&
          (i % (2*dimTab))==0) && !(i>=2*dimTab*dimTab))
    {
        if(*(laComplementaire+1)==0)
        {
            return false;
        }

        if(*(laColonneEnConstruction+i)+
               *(laColonneEnConstruction+i+1)== *(laComplementaire+1))
        {
            contact=true;
        }
        i+=2;
    }

    return contact;
}

bool Somme::testContactSup(int *laColonneEnConstruction,
                           int *laComplementaire, int dimTab)
{
    int i=0;
    while(*(laComplementaire+i)!=0)
    {
        i+=2*dimTab;
    }

    int positionDeContact=*(laComplementaire+i-2*dimTab)+
                          *(laComplementaire+i-2*dimTab+1);

    bool contact=false;

    int j=0;

    while(!(*(laColonneEnConstruction+j)==0 && (j % (2*dimTab))==0) &&
          !(j>=2*dimTab*dimTab))
    {
        if(*(laColonneEnConstruction+j+1)==positionDeContact)
        {
            contact=true;
        }
```

```
        j+=2;
    }

    return contact;
}

int *Somme::additionSansContact(int *laColonneEnConstruction,
                                int *laComplementaire, int dimTab)
{
    int gamma=
        getPositionDeLaColElemLaPlusPresSousLaComp(
                                                laColonneEnConstruction,
                                                *(laComplementaire+1),
                                                dimTab);

    int h=getNbrDeColElem(laComplementaire, dimTab);
    int l=getNbrDeColElem(laColonneEnConstruction, dimTab);

    for(int i=2*dimTab*(l)-1; i>=gamma*(2*dimTab)+2*dimTab; i--)
    {
        *(laColonneEnConstruction+i+2*dimTab*h)=
                                        *(laColonneEnConstruction+i);
    }

    for(int i=0; i<2*dimTab*h; i++)
    {
        *(laColonneEnConstruction+(gamma+1)*2*dimTab+i)=
                                            *(laComplementaire+i);
    }

    int *retour = new int[2*dimTab*dimTab];
    for(int i=0; i<2*dimTab*dimTab; i++)
    {
        *(retour +i)=*(laColonneEnConstruction + i);
    }

    return retour;
}

int *Somme::additionContactInf(int *laColonneEnConstruction,
                               int *laComplementaire, int dimTab)
{
    int positionCESAmodifier=
```

```
            getPositionDeLaCESAmodifier(laColonneEnConstruction,
                                        *(laComplementaire+1), dimTab);

    *(laColonneEnConstruction+positionCESAmodifier)+=
                                                    *(laComplementaire);

    int h=getNbrDeColElem(laComplementaire, dimTab);

    int i=0;
    while(i<2*dimTab*h)
    {
        *(laComplementaire+i)=*(laComplementaire+2*dimTab+i);
        i++;
    }

    return additionSansContact(laColonneEnConstruction,
                               laComplementaire, dimTab);
}

int *Somme::additionContactSup(int *laColonneEnConstruction,
                               int *laComplementaire, int dimTab)
{
    int h=getNbrDeColElem(laComplementaire, dimTab);

    int positionCESAmodifier=
        getPositionSUPDeLaCESAmodifier(
                                  laColonneEnConstruction,
                                  *(laComplementaire+(h-1)*2*dimTab+1)+
                                  *(laComplementaire+(h-1)*2*dimTab),
                                  dimTab);

    *(laColonneEnConstruction+positionCESAmodifier)+=
                                    *(laComplementaire+(h-1)*2*dimTab);

    *(laColonneEnConstruction+positionCESAmodifier+1)=
                                  *(laComplementaire+(h-1)*2*dimTab+1);

    int i=0;
    while(i<2*dimTab)
    {
        *(laComplementaire+2*dimTab*(h-1)+i)=0;
        i++;
    }
```

```
    return additionSansContact(laColonneEnConstruction,
                               laComplementaire, dimTab);
}

int *Somme::additionContactSupInf(int *laColonneEnConstruction,
                                  int *laComplementaire, int dimTab)
{
    int h=getNbrDeColElem(laComplementaire, dimTab);

    int positionCESAmodifier=
        getPositionSUPDeLaCESAmodifier(
                                  laColonneEnConstruction,
                                  *(laComplementaire+(h-1)*2*dimTab+1)+
                                  *(laComplementaire+(h-1)*2*dimTab),
                                  dimTab);

    *(laColonneEnConstruction+positionCESAmodifier)+=
                                    *(laComplementaire+(h-1)*2*dimTab);

    *(laColonneEnConstruction+positionCESAmodifier+1)=
                                  *(laComplementaire+(h-1)*2*dimTab+1);

    int i=0;
    while(i<2*dimTab)
    {
        *(laComplementaire+2*dimTab*(h-1)+i)=0;
        i++;
    }

    return additionContactInf(laColonneEnConstruction,
                              laComplementaire, dimTab);
}

int *Somme::additionFusion1(int *laColonneEnConstruction,
                            int *laComplementaire, int dimTab)
{
    int positionCESAmodifier=
                  getPositionDeLaCESAmodifier(laColonneEnConstruction,
                                              *(laComplementaire+1),
                                              dimTab);

    *(laColonneEnConstruction+positionCESAmodifier)+=
```

```
        *(laComplementaire)+
                    *(laColonneEnConstruction+positionCESAmodifier+2);

    int i=2;
    while(*(laColonneEnConstruction+positionCESAmodifier+i)!=0)
    {
        *(laColonneEnConstruction+positionCESAmodifier+i)=
                  *(laColonneEnConstruction+positionCESAmodifier+i+2);
        i++;
    }

    int *retour = new int[2*dimTab*dimTab];
    for(int i=0; i<2*dimTab*dimTab; i++)
    {
        *(retour +i)=*(laColonneEnConstruction + i);
    }

    return retour;
}

int *Somme::additionFusion2(int *laColonneEnConstruction,
                            int *laComplementaire, int dimTab)
{
    int positionSUPCESAmodifier=
                getPositionSUPDeLaCESAmodifier(laColonneEnConstruction,
                                               *(laComplementaire+1)+
                                               *(laComplementaire),
                                               dimTab);

    int positionINFCESAmodifier=
                   getPositionDeLaCESAmodifier(laColonneEnConstruction,
                                               *(laComplementaire+1),
                                               dimTab);

    *(laColonneEnConstruction+positionINFCESAmodifier)+=
        *(laComplementaire)+
                    *(laColonneEnConstruction+positionSUPCESAmodifier);

    int i=2;
    while(*(laColonneEnConstruction+positionSUPCESAmodifier+i)!=0)
    {
        *(laColonneEnConstruction+positionINFCESAmodifier+i)=
                  *(laColonneEnConstruction+positionSUPCESAmodifier+i);
```

```
        i++;
    }

    int j=0;
    while(!(*(laColonneEnConstruction+positionSUPCESAmodifier+j)==0 &&
          j % (2*dimTab) == 0))
    {
        *(laColonneEnConstruction+positionSUPCESAmodifier+j)=
        *(laColonneEnConstruction+positionSUPCESAmodifier+j+2*dimTab);
        j++;
    }

    for(int i=0; i<2*dimTab; i++)
    {
        *(laColonneEnConstruction+positionSUPCESAmodifier+j+i)=0;
    }

    int *retour = new int[2*dimTab*dimTab];
    for(int i=0; i<2*dimTab*dimTab; i++)
    {
        *(retour +i)=*(laColonneEnConstruction + i);
    }

    return retour;
}

int *Somme::additionFusion3(int *laColonneEnConstruction,
                            int *laComplementaire, int dimTab)
{
    int positionSUPCESAmodifier=
                getPositionSUPDeLaCESAmodifier(laColonneEnConstruction,
                                               *(laComplementaire+1)+
                                               *(laComplementaire),
                                               dimTab);

    int positionINFCESAmodifier=
                   getPositionDeLaCESAmodifier(laColonneEnConstruction,
                                               *(laComplementaire+1),
                                               dimTab);

    *(laColonneEnConstruction+positionINFCESAmodifier)+=
        *(laComplementaire)+
                    *(laColonneEnConstruction+positionSUPCESAmodifier);
```

```
int tailleColElemAInserer=getTailleColElem(laColonneEnConstruction,
                                           positionSUPCESAmodifier);

int tailleResteColElemATranslater=
                          getTailleColElem(laColonneEnConstruction,
                                           positionINFCESAmodifier);

for(int i=positionINFCESAmodifier+
             2*tailleResteColElemATranslater+1;
    i>positionINFCESAmodifier+1;
    i--)
{
    *(laColonneEnConstruction+i+2*tailleColElemAInserer)=
                                        *(laColonneEnConstruction+i);
}

int k=2;
while(*(laColonneEnConstruction+positionSUPCESAmodifier+k)!=0)
{
    *(laColonneEnConstruction+positionINFCESAmodifier+k)=
             *(laColonneEnConstruction+positionSUPCESAmodifier+k);
    k++;
}

int j=0;
while(!(*(laColonneEnConstruction+positionSUPCESAmodifier+j)==0 &&
      j % (2*dimTab) == 0))
{
    *(laColonneEnConstruction+positionSUPCESAmodifier+j)=
    *(laColonneEnConstruction+positionSUPCESAmodifier+j+2*dimTab);
    j++;
}

for(int i=0; i<2*dimTab; i++)
{
    *(laColonneEnConstruction+positionSUPCESAmodifier+j+i)=0;
}

int *retour = new int[2*dimTab*dimTab];
for(int i=0; i<2*dimTab*dimTab; i++)
{
    *(retour +i)=*(laColonneEnConstruction + i);
```

```
    }

    return retour;
}

int *Somme::additionFusion4(int *laColonneEnConstruction,
                            int *laComplementaire, int dimTab)
{
    int positionSUPCESAmodifier=
                getPositionSUPDeLaCESAmodifier(laColonneEnConstruction,
                                               *(laComplementaire+1)+
                                               *(laComplementaire),
                                               dimTab);

    int positionINFCESAmodifier=
                  getPositionDeLaCESAmodifier(laColonneEnConstruction,
                                               *(laComplementaire+1),
                                               dimTab);

    *(laColonneEnConstruction+positionINFCESAmodifier)+=
        *(laComplementaire)+
                    *(laColonneEnConstruction+positionSUPCESAmodifier);

    int tailleColElemAInserer=positionINFCESAmodifier % (2*dimTab)+2;

    int tailleResteColElemATranslater=
                            getTailleColElem(laColonneEnConstruction,
                                             positionSUPCESAmodifier);

    for(int i=positionSUPCESAmodifier+
                  2*tailleResteColElemATranslater+1;
        i>=positionSUPCESAmodifier;
        i--)
    {
        *(laColonneEnConstruction+i+tailleColElemAInserer-2)=
                                          *(laColonneEnConstruction+i);
    }

    int k=0;

    int debutColElemINF=
                      positionINFCESAmodifier-tailleColElemAInserer+2;
```

```
    while(*(laColonneEnConstruction+debutColElemINF+k)!=0)
    {
        *(laColonneEnConstruction+positionSUPCESAmodifier+k)=
                        *(laColonneEnConstruction+debutColElemINF+k);
        k++;
    }

    int j=0;
    while(!(*(laColonneEnConstruction+debutColElemINF+j)==0 &&
          j % (2*dimTab) == 0))
    {
        *(laColonneEnConstruction+debutColElemINF+j)=
                 *(laColonneEnConstruction+debutColElemINF+j+2*dimTab);
        j++;
    }

    int *retour = new int[2*dimTab*dimTab];
    for(int i=0; i<2*dimTab*dimTab; i++)
    {
        *(retour +i)=*(laColonneEnConstruction + i);
    }

    return retour;
}

int *Somme::additionFusion(int *laColonneEnConstruction,
                           int *laComplementaire, int dimTab)
{
    int positionSUPCESAmodifier=
                getPositionSUPDeLaCESAmodifier(laColonneEnConstruction,
                                               *(laComplementaire+1)+
                                               *(laComplementaire),
                                               dimTab);

    int positionINFCESAmodifier=
                   getPositionDeLaCESAmodifier(laColonneEnConstruction,
                                               *(laComplementaire+1),
                                               dimTab);

    if(positionSUPCESAmodifier==positionINFCESAmodifier+2)
    {
        return additionFusion1(laColonneEnConstruction,
                               laComplementaire, dimTab);
```

```
    }

    if(*(laColonneEnConstruction+positionINFCESAmodifier+2)==0 &&
       *(laColonneEnConstruction+positionSUPCESAmodifier-1)==0)
    {
        return additionFusion2(laColonneEnConstruction,
                               laComplementaire, dimTab);
    }

    if(*(laColonneEnConstruction+positionINFCESAmodifier+2)!=0 &&
       *(laColonneEnConstruction+positionSUPCESAmodifier-1)==0)
    {
        return additionFusion3(laColonneEnConstruction,
                               laComplementaire, dimTab);
    }

    if(*(laColonneEnConstruction+positionINFCESAmodifier+2)==0 &&
       *(laColonneEnConstruction+positionSUPCESAmodifier-1)!=0)
    {
        return additionFusion4(laColonneEnConstruction,
                               laComplementaire, dimTab);
    }

    int *retour = new int[2*dimTab*dimTab];
    for(int i=0; i<2*dimTab*dimTab; i++)
    {
        *(retour +i)=*(laColonneEnConstruction + i);
    }

    return retour;
}

int Somme::getTailleColElem(int *laColonneEnConstruction,
                            int positionCESAmodifier)
{
    int taille=0;
    while(*(laColonneEnConstruction+positionCESAmodifier+2*taille)!=0)
    {
        taille++;
    }

    return taille-1;
}
```

```
int Somme::getPositionDeLaCESAmodifier(int *laColonneEnConstruction,
                                       int laPosition, int dimTab)
{
    int positionAretourner=0;
    while(*(laColonneEnConstruction+positionAretourner)+
          *(laColonneEnConstruction+positionAretourner+1)!=laPosition)
    {
        positionAretourner+=2;
    }

    return positionAretourner;
}

int Somme::getPositionSUPDeLaCESAmodifier(int *laColonneEnConstruction,
                                          int laPosition, int dimTab)
{
    int positionAretourner=0;
    while(*(laColonneEnConstruction+positionAretourner+1)!=laPosition)
    {
        positionAretourner+=2;
    }

    return positionAretourner;
}


int Somme::getPositionDeLaColElemLaPlusPresSousLaComp(
                int *laColEnConstruction, int positionInf, int dimTab)
{
    int positionAretourner=0;

    while(*(laColEnConstruction+positionAretourner*2*dimTab+1)<
                                                        positionInf &&
          *(laColEnConstruction+positionAretourner*2*dimTab+1)!=0)
    {
        positionAretourner++;
    }

    return positionAretourner-1;
}

int Somme::getNbrDeColElem(int *laColonne, int dimTab)
```

```
{
    int Aretourner=0;
    while(Aretourner<=dimTab && *(laColonne+2*dimTab*Aretourner)!=0)
    {
        Aretourner++;
    }

    return Aretourner;
}

//************************* Fin du fichier *************************//
```

ANNEXE 2

Une formule itérative

Les formules qui suivent servent à énumérer les polyominos de volume minimal inscrits dans un prisme rectangulaire. Certains détails dans les explications de ces formules ont été omis volontairement pour alléger le texte. Globalement, en rassemblant ces formules, on en obtient une qui renferme plusieurs sommations parfois irréductibles. Ces formules nous ont permis de vérifier certains résultats numériquement pour des prismes de dimensions allant jusqu'à $50 \times 50 \times 50$.

n	Nombre de polyominos minimaux inscrits dans un prisme $n \times n \times n$
1	1
2	32
3	2401
4	87056
5	2256145
6	52177880
7	1188699457
8	27521161352
9	650038133833
10	15611269838480
20	153615254677439075108608904O
30	207876117575048544952609239759402838711256
40	3187491275820440290345967010029089308973827550321058554 4
50	52282273582754212098881768212703020411593167305435599082008341527044 80

Tableau IV Nombre de polyominos de volume minimal dans un prisme $n \times n \times n$

Dans ce tableau, on peut voir que le nombre de décimales en fonction de n augmente de façon constante pour ces premières valeurs de n, mais rien jusqu'ici ne permet de trouver rigoureusement des résultats sur le comportement asymptotique du nombre de ces polyominos en fonction de la taille n du prisme cubique.

Le nombre de polyominos d'aire minimale inscrits dans un rectangle $b \times h$ est donné par

$$p_{min}(b,h) = \begin{cases} 8\begin{pmatrix} h+b-2 \\ h-1 \end{pmatrix} - 6 - bh - 2(b-1)(h-1) \ \texttt{si} \ b,h>1, \\ 1 \text{ si } b=1 \text{ ou } h=1. \end{cases}$$

On considérera les polyominos d'aire minimale qui ont exactement deux cellules chacune en contact avec trois cellules. Ces polyominos à 4 feuilles n'occupent aucun coin du rectangle $b \times h$ dans lequel ils sont inscrits et il y en a

$$\begin{aligned} p'_{min}(b,h) &= 2\sum_{i=2}^{r-1}\sum_{j=2}^{h-1}\sum_{m=i}^{b-1}\sum_{n=j}^{h-1}\begin{pmatrix} m-i+n-j \\ m-i \end{pmatrix} - (b-2)(h-2) \\ &= 2\,\frac{(h-2+b)\,(h-3+b)\begin{pmatrix} b-4+h \\ b-2 \end{pmatrix}}{(h-1)\,(b-1)} - 3\,hb + 4\,b + 4\,h - 8. \end{aligned}$$

Le nombre de polyominos d'aire minimale occupant un coin fixé dans un rectangle $b \times k$ est

$$\begin{aligned} N_{1coin}(b,k) &= \sum_{i=2}^{b}\sum_{j=2}^{k}\begin{pmatrix} b+k-i-j \\ b-i \end{pmatrix} + \begin{pmatrix} k+b-2 \\ b-1 \end{pmatrix} \\ &= 2\begin{pmatrix} k+b-2 \\ b-1 \end{pmatrix} - 1. \end{aligned}$$

Définition 2.0.1 Le ***degré*** d'une cellule est égal au nombre de cellules qui sont en contact avec elle.

Définition 2.0.2 Dans un polyomino d'aire minimale, une ***branche*** est un ensemble connexe de cellules reliant une cellule de degré 3 ou 4 à un côté du rectangle circonscrit.

Définition 2.0.3 Dans un polyomino d'aire minimale, un ***tronc*** est un ensemble de cellules reliant deux cellules de degré 3.

Dans cette annexe, les parties de polyominos 3D qui seront construites seront souvent vues comme des constructions 2D sous les branches et le tronc de la projection d'un polyomino 3D. Dans le cas où la partie de la projection sous laquelle se trouve une partie de polyomino 3D qui nous intéresse ne sera pas sur une même droite, on imaginera que ce polyomino est « déplié » comme à l'illustration 36.

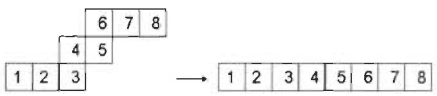

Figure 36 Le « dépliage » d'une partie de polyomino 3D vu du haut

Les variables l et q seront réservées pour représenter la hauteur des cellules sous une branche où ces cellules sont toutes à la même hauteur. Généralement, les polyominos seront découpés en 3 parties : les cellules dont la hauteur est plus petite ou égale à l, les cellules dont la hauteur est entre l et q puis les cellules dont la hauteur est plus grande ou égale à q. La partie entre l et q sera sous le tronc ; les deux autres parties seront sous les branches.

Si la projection d'un polyomino de volume minimal de hauteur h a trois branches et que les cellules de ce polyomino ne sont pas toutes à la même hauteur sous exactement

deux branches de longueurs b_1 et b_2, alors il y a

$$
\begin{aligned}
N_{2branches}(b_1,b_2,h) = \sum_{l=2}^{h-1} & \left[\left(N_{1coin}\left(b_1+1,l\right)-l\right)\left(N_{1coin}\left(b_2+1,h-l+1\right)-h+l-1\right)\right. \\
& \left.+\left(N_{1coin}\left(b_1+1,h-l+1\right)-h+l-1\right)\left(N_{1coin}\left(b_2+1,l\right)-l\right)\right] \\
= & -\frac{1}{3(h-2)}\left[(48+36b_2+12h)\begin{pmatrix} h-1+b_2 \\ b_2+2 \end{pmatrix}+(-48+24h)\begin{pmatrix} b_1+h-1 \\ b_1 \end{pmatrix}\right] \\
& -\frac{1}{3(h-2)}\left[(-48+24h)\begin{pmatrix} h-1+b_2 \\ b_2 \end{pmatrix}+(36b_1+12h+48)\begin{pmatrix} b_1+h-1 \\ 2+b_1 \end{pmatrix}\right] \\
& -\frac{1}{3(h-2)}\left[((-24h+48)\begin{pmatrix} h+b_2+b_1 \\ h-1 \end{pmatrix}-7h^3+4h^2+28h-h^4\right]
\end{aligned}
$$

façons de construire ce polyomino où l est la hauteur des cellules sous la troisième branche (ni b_1 ni b_2). Considérant que la cellule de degré 3 est à la position (i,j), que le polyomino à trois branche est vu comme un polyomino coin occupant le coin supérieur gauche du rectangle avant rotation et que la branche où les cellules sont toutes à la même hauteur peut être l'une des trois branches de longueur $j-1, b-i$ et $i+k-j-1$,

on a que

$$
\begin{aligned}
N_{1hauteur}(b,k,h) = 4\sum_{i=2}^{b-1}\sum_{j=2}^{k-1}\begin{pmatrix}(k-j)+i-1\\ i-1\end{pmatrix}[&N_{2branches}(j-1,i+k-j-1,h)\\
&+N_{2branches}(b-i,i+k-j-1,h)\\
&+N_{2branches}(b-i,j-1,h)]\\
&+2\sum_{i=2}^{b-1}[N_{2branches}(k-1,i-1,h)\\
&+N_{2branches}(b-i,i-1,h)\\
&+N_{2branches}(b-i,k-1,h)]\\
&+2\sum_{j=2}^{k-1}[N_{2branches}(j-1,k-j,h)\\
&+N_{2branches}(b-1,k-j,h)\\
&+N_{2branches}(b-1,j-1,h)]
\end{aligned}
$$

est égal au nombre de polyominos de volume minimal inscrits dans un prisme $b \times k \times h$ avec des cellules toutes à la même hauteur sous exactement une branche et dont la projection sur la face supérieure du prisme a exactement trois branches.

On a que le nombre de polyominos de volume minimal de hauteur h avec deux branches pour chacune desquelles la position verticale des cellules est constante sous un polyomino fixé d'aire minimale à trois branches avec, dans la troisième branche de

longueur $b-1$, des cellules qui ne sont pas toutes à la même hauteur est

$$\begin{aligned} N_{1branche}(b,h) &= 2(2\sum_{l=2}^{h-1}(h-l)(N_{1coin}(b,l)-l)+\sum_{l=2}^{h-1}(N_{1coin}(b,l)-l)) \\ &\quad + p_{min}(b,h)-h \\ &= 2\sum_{l=2}^{h-1}(2(h-l)+1)(N_{1coin}(b,l)-l)+p_{min}(b,h)-h. \end{aligned}$$

La branche sous laquelle les cellules ne sont pas toutes à la même hauteur peut être l'une des trois branches du polyomino d'aire minimale sous lequel on construit un polyomino de volume minimal. Alors,

$$\begin{aligned} N_{2hauteurs}(b,k,h) &= 4\sum_{i=2}^{b-1}\sum_{j=2}^{k-1}\begin{pmatrix}(k-j)+i-1\\ i-1\end{pmatrix}[N_{1branche}(j,h) \\ &\quad + N_{1branche}(i+k-j,h) \\ &\quad + N_{1branche}(b-i+1,h)] \\ &\quad + 2\sum_{i=2}^{b-1}[N_{1branche}(k,h) \\ &\quad + N_{1branche}(b-i+1,h) \\ &\quad + N_{1branche}(i,h)] \\ &\quad + 2\sum_{j=2}^{k-1}[N_{1branche}(j,h) \\ &\quad + N_{1branche}(k-j+1,h) \\ &\quad + N_{1branche}(b,h)] \end{aligned}$$

est égal au nombre de polyominos de volume minimal sous un polyomino d'aire minimale à trois branches avec 2 branches sous lesquelles les cellules sont à hauteur fixée.

$$N_{3hauteurs}(b,k,h) = 4\sum_{i=2}^{b-1}\sum_{j=2}^{k-1}\begin{pmatrix}(k-j)+i-1\\ i-1\end{pmatrix}h^3 + 2\sum_{i=2}^{b-1}h^3 + 2\sum_{j=2}^{k-1}h^3$$
$$= 4h^3\sum_{i=2}^{b-1}\sum_{j=2}^{k-1}\begin{pmatrix}(k-j)+i-1\\ i-1\end{pmatrix} + h^3(2(b-2)+2(k-2))$$

est le nombre de polyominos de volume minimal à trois branches à hauteur des cellules fixée sous les polyominos d'aire minimale à trois branches.

Le nombre de polyominos de volume minimal sous un polyomino d'aire minimale à trois branches ou occupant deux coins opposés est donc

$$N_{p3}(b,k,h) = N_{1hauteur}(b,k,h) + N_{2hauteurs}(b,k,h) + N_{3hauteurs}(b,k,h) +$$
$$2\begin{pmatrix}b+k-2\\ b-1\end{pmatrix}p_{min}(b+k-1,h).$$

Soit l'ensemble P_4 des polyominos de volume minimal dont la projection sur la face supérieure du prisme dans lequel il sont inscrits est un polyomino n'occupant aucun coin et ayant deux cellules de degré 3. Cette projection a 4 branches.

Soit les notations de la figure 37.

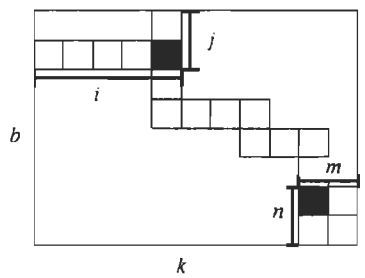

Figure 37 Positions des cellules en contact avec trois cellules

Dans P_4, pour des entiers i, j, m, n fixés, les polyominos où les cellules sous le tronc et sous les branches de longueur $i-1$ et $j-1$ sont toutes à la même hauteur entre les cellules sous les branches de longueur $m-1$ et $n-1$ avec des cellules qui ne sont pas à la même hauteur strictement sous l'une de ces deux dernières branches sont énumérés par

$$troncEntreAutreCel(h,m,n) = 2\sum_{l=2}^{h-1}(N_{1coin}(m,l) + N_{1coin}(n,l) - 2l)(h-l)$$

où h est la hauteur du prisme circonscrit.

Dans P_4, pour des entiers i, j, m, n fixés, les polyominos dont les cellules sous le tronc et deux branches à la même extrémité de ce tronc sont à une même hauteur entre les positions verticales des cellules sous les deux autres branches avec, sous chacune de ces deux autres branches, les cellules toutes à la même hauteur sont énumérés par

$$troncEntre2(h) = 2\sum_{l=1}^{h}\sum_{q=l+2}^{h}(q-l-1)$$

La longueur du tronc est $r = b - i - m + k - j - n + 3$.

Les polyominos de P_4 dont les cellules sous la branche de longueur $i-1$ sont à une même hauteur comme sous la branche de longueur $j-1$ sans que, d'une branche à l'autre, les cellules soient nécessairement à une seule hauteur sont énumérés par

$$N_{40branche}(b,h,k,i,j,m,n) = \begin{pmatrix} r-1 \\ b-i-m+1 \end{pmatrix}$$
$$* \left[2\sum_{l=1}^{h}[(2l-1)(N_{1coin}(r,h-l+1) - \begin{pmatrix} r-1+h-l \\ h-l \end{pmatrix}$$
$$+ \sum_{q=l}^{h} \begin{pmatrix} q-l+r-1 \\ q-l \end{pmatrix} * (N_{1coin}(m,h-q+1) + N_{1coin}(n,h-q+1) - 1))$$
$$+ (N_{1coin}(r,l) - \begin{pmatrix} r-1+l-1 \\ l-1 \end{pmatrix})$$
$$* (N_{1coin}(m,h-l+1) + N_{1coin}(n,h-l+1) - 1)]$$
$$- 2h - 2(r-2) + 2troncEntre2(h) + troncEntreAutreCel(h,m,n)$$
$$+ N_{2branches}(m-1,n-1,h) + p'_{min}(r,h)] .$$

Dans P_4, pour des entiers i,j,m,n fixés, les polyominos qui n'ont pas les cellules à la même hauteur sous exactement une des branches de longueur $i-1$ ou $j-1$ sans que ces cellules ne touchent à la fois le haut et le bas du prisme sous ces branches est donné par

$$N_{41branche}(b,h,k,i,j,m,n) = 2 \begin{pmatrix} r-1 \\ b-i-m+1 \end{pmatrix}$$
$$* \sum_{l=2}^{h} \left[(N_{1coin}(j,l) + N_{1coin}(i,l) - 2l)(N_{1coin}(r,h-l+1) - \begin{pmatrix} r-1+h-l \\ h-l \end{pmatrix} \right.$$
$$+ \sum_{q=l}^{h} \begin{pmatrix} q-l+r-1 \\ q-l \end{pmatrix}$$
$$\left. * (N_{1coin}(m,h-q+1) + N_{1coin}(n,h-q+1) - 1) + (h-l)) \right]$$

Dans P_4, pour des entiers i, j, m, n fixés, les polyominos pour lesquels les cellules sous les branches de longueur $i-1$ et $j-1$ ne sont pas à la même hauteur sont énumérés par

$$N_{42branches}(b,h,k,i,j,m,n) = \begin{pmatrix} r-1 \\ b-i-m+1 \end{pmatrix} * N_{2branches}(i-1, j-1, h)$$

Dans P_4, pour des entiers i, j, m, n fixés, les polyominos qui touchent le haut et le bas du prisme sous une même branche sont énumérés par

$$\begin{aligned} N_{memeBranche}(b,h,k,i,j,m,n) = & \begin{pmatrix} r-1 \\ b-i-m+1 \end{pmatrix} \\ & * (p_{min}(i,h) - 2N_{1coin}(i,h) + h \\ & + p_{min}(j,h) - 2N_{1coin}(j,h) + h \\ & + p_{min}(m,h) - 2N_{1coin}(m,h) + h \\ & + p_{min}(n,h) - 2N_{1coin}(n,h) + h) \end{aligned}$$

Dans P_4, pour des entiers i, j, m, n fixés, le nombre de polyominos est donné par

$$\begin{aligned} N_4(b,h,k,i,j,m,n) = & N_{40branche}(b,h,k,i,j,m,n) \\ & + N_{41branche}(b,h,k,i,j,m,n) \\ & + N_{42branches}(b,h,k,i,j,m,n) \\ & + N_{memeBranche}(b,h,k,i,j,m,n). \end{aligned}$$

Soit P_X, l'ensemble des polyominos de volume minimal dont la projection sur la face supérieure du prisme est un polyomino croix.

Dans P_X, les polyominos dont les cellules ne sont pas à la même hauteur sous chacune de deux branches opposées (suivant une même droite) sont énumérés par

$$\begin{aligned} X_{branchesOp}(b,h,k) = \sum_{j=2}^{k-1}\sum_{l=2}^{h-1}[&(b-2)X_{2branchesOp}(h,k,l,l,j) \\ &+ \sum_{q=l+1}^{h-1}[(b-2)2X_{2branchesOp}(h,k,q,l,j)]] \\ + \sum_{i=2}^{b-1}\sum_{l=2}^{h-1}[&(k-2)X_{2branchesOp}(h,b,l,l,i) \\ &+ \sum_{q=l+1}^{h-1}[(k-2)2X_{2branchesOp}(h,b,q,l,i)]] \end{aligned}$$

où

$$\begin{aligned} X_{2branchesOp}(h,k,q,l,b_1) = ((&N_{1coin}(b_1,l)-l) \\ &* (N_{1coin}(k-b_1+1,h-q+1)-(h-q+1)) \\ &+ (N_{1coin}(k-b_1+1,l)-l) \\ &\quad * (N_{1coin}(b_1,h-q+1)-(h-q+1))). \end{aligned}$$

Dans P_X, les polyominos dont les cellules ne sont pas à la même hauteur sous chacune de deux branches consécutives (selon un parcourt dans le sens horaire) sont énumérés par

$$X_{branchesC}(b,h,k) = \sum_{i=2}^{b-1}\sum_{j=2}^{k-1}\sum_{l=2}^{h-1}[4((N_{1coin}(i,l)-l)(N_{1coin}(j,h-l+1)-(h-l+1))$$
$$+ (N_{1coin}(j,l)-l)(N_{1coin}(i,h-l+1)-(h-l+1)))$$
$$+ 4\sum_{q=l+1}^{h-1} 2((N_{1coin}(i,l)-l)(N_{1coin}(j,h-q+1)-(h-q+1))$$
$$+ (N_{1coin}(j,l)-l)(N_{1coin}(i,h-q+1)-(h-q+1)))].$$

Dans P_X, les polyominos dont les cellules ne sont pas à la même hauteur sous exactement une branche sans que celles-ci ne touchent le haut et le bas du prisme sous cette branche sont énumérés par

$$X_{1branche}(b,h,k) = \sum_{i=2}^{b-1}\sum_{l=1}^{h} 2(k-2)((N_{1coin}(i,l)-l)(3(h-l+1)^2-3(h-l+1)+1)$$
$$+ (N_{1coin}(i,h-l+1)-(h-l+1))(3l^2-3l+1))$$
$$+ \sum_{j=2}^{k-1}\sum_{l=1}^{h} 2(b-2)((N_{1coin}(j,l)-l)(3(h-l+1)^2-3(h-l+1)+1)$$
$$+ (N_{1coin}(j,h-l+1)-(h-l+1))(3l^2-3l+1)).$$

Dans P_X, les polyominos dont les cellules touchent le haut et le bas du prisme sous une même branche sont énumérés par

$$X_{hautBas}(b,h,k) = \sum_{i=2}^{b-1} 2(k-2)(p_{min}(i,h)-2N_{1coin}(i,h)+2+(h-2))$$
$$+ \sum_{j=2}^{k-1} 2(b-2)(p_{min}(j,h)-2N_{1coin}(j,h)+2+(h-2)).$$

Enfin, les polyominos de volume minimal inscrits dans un prisme $b \times h \times k$ sont énumérés par

$$\begin{aligned} N_{polyMin}(b,h,k) = 2\sum_{i=2}^{b-1}\sum_{j=2}^{k-1}\Bigg[\sum_{m=2}^{b-i}\sum_{n=2}^{k-j+1} N_4(b,h,k,i,j,m,n) \\ + \sum_{n=2}^{k-j} N_4(b,h,k,i,j,b-i+1,n)\Bigg] \\ + X_{branchesOp}(b,h,k) + X_{branchesC}(b,h,k) + X_{1branche}(b,h,k) \\ + X_{hautBas}(b,h,k) + (b-2)(k-2)h^4 + N_{p3}(b,k,h). \end{aligned}$$

ANNEXE 3

Un programme utilisant des filtres

Le programme qui suit débute avec toutes les constructions ayant $b+k+h-2$ cellules dans un prisme de dimensions $b \times k \times h$. Ces constructions sont possiblement non connexes. Ensuite, des filtres sont utilisés pour éliminer certaines de ces constructions avant de faire sur elles un test de connexité pour ne conserver que les polyominos de volume minimal. Ceux-ci sont affichés dans un fichier texte. Ce programme peut être modifié pour compter des polyominos inscrits dans un prisme $b \times k \times h$ ayant un volume non minimal.

```
//*********************************************************************//
// Point d'entrée du programme: méthode main()                         //
//*********************************************************************//

#include "Traitement.h"
#include <fstream>
#include <cmath>
#include <string>
#include <time.h>
#include <sstream>
#include <iostream>
using namespace std;

int main()
{
    std::cout<<"Cette application d\202termine le nombre de polyominos
               tridimensionnels \nde volume minimal inscrits dans un
```

```
            prisme rectangulaire donn\202.";

//Prise des données...
std::cout<<"\n\nEntrer la longueur du prisme rectangulaire dans
            lequel les polyominos sont \ninscrits. (Le plus grand
            prisme)";
int longueurDuPrisme;
std::cin>>longueurDuPrisme;

std::cout<<"\nEntrer la hauteur du prisme.";
int hauteurDuPrisme;
std::cin>>hauteurDuPrisme;

std::cout<<"\nEntrer l'\202paisseur du prisme.";
int epaisseurDuPrisme;
std::cin>>epaisseurDuPrisme;
//Fin de la prise de données

std::cout<<"\nLes polyominos peuvent être affich\202s dans un
            fichier texte. Entrez le nom du fichier.";
ofstream fichier_res;

string nom_fichier;
std::cin>> nom_fichier;
nom_fichier.append(".txt");

//Traitement des données...
std::cout<<"\n\n\nLe processus est en cours...\n\n";
```

```
    std::cout<<"Pour ce prisme, il y a "
             << Traitement::principale(longueurDuPrisme,
                                       hauteurDuPrisme,
                                       epaisseurDuPrisme, nom_fichier)
             <<" polyominos de volume minimal.";

    //pour fermer l'application...
    std::cout << "\n\n\nEntrer un caract\212re pour terminer.";
    char fin;
    std::cin >> fin;
    return 0;
}


//************************** Fin du fichier ************************//

//*********************************************************************//
// Classe: Traitement                                                  //
// Tout le traitement se fait dans cette classe où plusieurs filtres //
// sont appliqués avant de faire un test de connexité. Les polyominos//
// retenus sont affichés dans un fichier texte.                        //
//*********************************************************************//

#include "Traitement.h"
#include <fstream>
#include <cmath>
#include <string>
#include <time.h>
#include <sstream>
```

```
#include <iostream>
using namespace std;

ofstream ecriture_fichier;
int compteurDePolyominos;

Traitement::Traitement(void)
{
}

Traitement::~Traitement(void)
{
}

double Traitement::principale(int l, int h, int e, string nom_fichier)
{
    ecriture_fichier.open(nom_fichier.c_str());
    compteurDePolyominos = 1;

    int *premierDuPaquet=premiereConstruction(l+h+e-2, l*h*e);

    double nbrPoly=0;

    while(!estFinDeListe(premierDuPaquet))
    {
        int *premierDuPaqSuivant_nbrPoly=
                    faireConstructionsMin(l,h,e,
                                          premierDuPaquet);
```

```
        nbrPoly += *(premierDuPaqSuivant_nbrPoly + l*h*e+1);
        premierDuPaquet=copierTabBin(premierDuPaqSuivant_nbrPoly,
                                     l*h*e);

        delete[] premierDuPaqSuivant_nbrPoly;
    }

    return nbrPoly;
}

////////////////////////////////////////////////////////////////////////
// Les procédures qui suivent servent à faire la liste des construc-
// tions de volume minimal. Cette liste correspond aux manières de
// choisir l+h+e-2 éléments parmi l*h*e.
////////////////////////////////////////////////////////////////////////

int *Traitement::faireConstructionsMin(int l, int h, int e,
                                       int *premierDuPaquet)
{
    int volumePrisme=l*h*e;
    int volumeMin=l+h+e-2;

    int **listeDesConstructions=new int*[1000000];
    *(listeDesConstructions)=premierDuPaquet;

    int dernieri = 0;
    for(int i=1;
        i< 1000000 && !estFinDeListe(*(listeDesConstructions + i-1));
        i++)
```

```
{
    *(listeDesConstructions+i)=
                constructionSuivante(*(listeDesConstructions+i-1),
                                        volumePrisme, volumeMin);
    dernieri = i;
}

dernieri++;

int nbrPoly=calculPolyDansListe(listeDesConstructions, l, h, e);

int *premierDuPaqSuivant_nbrPoly=new int[volumePrisme+2];
if(estFinDeListe(*(listeDesConstructions + dernieri-1)))
{
    *(premierDuPaqSuivant_nbrPoly+1)=-1;
    *(premierDuPaqSuivant_nbrPoly + volumePrisme + 1)=nbrPoly;

    for(int j=0; j<dernieri; j++)
    {
        delete[] *(listeDesConstructions+j);
    }

    delete[] listeDesConstructions;

    return premierDuPaqSuivant_nbrPoly;
}

premierDuPaqSuivant_nbrPoly=
         constructionSuivante(*(listeDesConstructions +dernieri-1),
```

```
                                    volumePrisme, volumeMin);

    *(premierDuPaqSuivant_nbrPoly + volumePrisme + 1)=nbrPoly;

    for(int i=0; i<1000000; i++)
    {
        delete[] *(listeDesConstructions+i);
    }

    delete[] listeDesConstructions;

    return premierDuPaqSuivant_nbrPoly;
}

int *Traitement::premiereConstruction(int volMin, int volPrisme)
{
    int *constructionBin=new int[volPrisme+2];
    for(int i=1; i <= volMin; i++)
    {
        *(constructionBin+i)=1;
    }
    for(int i=volMin+1; i<=volPrisme; i++)
    {
        *(constructionBin+i)=0;
    }

    return constructionBin;
}
```

```
int *Traitement::constructionSuivante(int *constructionBin,
                                      int volPrisme, int volMin)
{
    int *constResultat= new int[volPrisme+2];

    for(int i=1; i<=volPrisme; i++)
    {
        *(constResultat+i)=*(constructionBin+i);
    }

    int nombreDeBitsADeplacer=getNombreDeBits(constResultat,
                                              volPrisme);

    if(nombreDeBitsADeplacer == volMin +1)
    {
        *(constResultat+1)=-1;
        return constResultat;
    }

    int positionBitADeplacer=getPositionBit(constResultat, volPrisme);

    for(int i=volPrisme-nombreDeBitsADeplacer+1; i<=volPrisme; i++)
    {
        *(constResultat+i)=0;
    }

    *(constResultat+positionBitADeplacer)=0;

    for(int i=positionBitADeplacer+1;
```

```
        i<=positionBitADeplacer+nombreDeBitsADeplacer;
        i++)
    {
        *(constResultat+i)=1;
    }

    return constResultat;
}

int Traitement::getNombreDeBits(int *tableauBin, int volPrisme)
{
    int nbrDeBits=0;
    int i=volPrisme;

    while(*(tableauBin+i)!=0)
    {
        nbrDeBits++;
        i--;
    }

    return nbrDeBits+1;
}

int Traitement::getPositionBit(int *constResultat, int volPrisme)
{
    int i=volPrisme;
    while(*(constResultat+i)==1)
    {
        i--;
```

```
    }
    while(*(constResultat+i)==0)
    {
        i--;
    }

    return i;
}

bool Traitement::estFinDeListe(int *TabBin)
{
    if(*(TabBin+1)==-1)
    {
        return true;
    }

    return false;
}

////////////////////////////////////////////////////////////////////////
// Traitement des blocs de tableaux binaires
////////////////////////////////////////////////////////////////////////

int Traitement::calculPolyDansListe(int **listeDesConstructions, int l,
                                    int h, int e)
{
    filtreXY(listeDesConstructions, l,h,e);
    filtreXZ(listeDesConstructions, l,h,e);
    filtreYZ(listeDesConstructions, l,h,e);
```

```
filtreX(listeDesConstructions, l,h,e);
filtreY(listeDesConstructions, l,h,e);
filtreZ(listeDesConstructions, l,h,e);

filtreConnexe(listeDesConstructions, l,h,e);

int nbrPoly=0;
int i=0;

while(i<1000000 && !estFinDeListe(*(listeDesConstructions+i)))
{

    if(*(*(listeDesConstructions+i))!=-1)
    {
        nbrPoly++;
        ecriture_fichier << compteurDePolyominos <<"\n";

        compteurDePolyominos++;
        for(int m=1; m<=h; m++)
        {
            for(int n=1; n<=e; n++)
            {
            for(int j=1; j<=l; j++)
                ecriture_fichier <<
                *(*(listeDesConstructions+i)+l*e*(m-1)+l*(n-1)+j);

            ecriture_fichier << " ";
            }
```

```
                ecriture_fichier << "\n";
            }
            ecriture_fichier << "\n\n";
        }
        i++;
    }

    return nbrPoly;
}

void Traitement::filtreXY(int **listeDesConstructions, int l, int h,
                          int e)
{
    for(int i=0;
        i<1000000 && !estFinDeListe(*(listeDesConstructions+i));
        i++)
    {
        if(!uneCelluleParPlaqueXY(*(listeDesConstructions+i), l, h, e))
        {
            *(*(listeDesConstructions+i))=-1;
        }
        else
        {
            *(*(listeDesConstructions+i))=0;
        }
    }
}

void Traitement::filtreXZ(int **listeDesConstructions, int l, int h,
```

```
                        int e)
{
    for(int i=0;
        i<1000000 && !estFinDeListe(*(listeDesConstructions+i));
        i++)
    {
        if(!uneCelluleParPlaqueXZ(*(listeDesConstructions+i), l, h, e))
        {
            *(*(listeDesConstructions+i))=-1;
        }
    }
}

void Traitement::filtreYZ(int **listeDesConstructions, int l, int h,
                        int e)
{
    for(int i=0;
    i<1000000 && !estFinDeListe(*(listeDesConstructions+i));
    i++)
    {
        if(!uneCelluleParPlaqueYZ(*(listeDesConstructions+i), l, h, e))
        {
            *(*(listeDesConstructions+i))=-1;
        }
    }
}

void Traitement::filtreX(int **listeDesConstructions,
                        int l,int h,int e)
```

```
{
    for(int i=0;
        i<1000000 && !estFinDeListe(*(listeDesConstructions+i));
        i++)
    {
        if(*(*(listeDesConstructions+i))==0)
        {
            if(!alignementDeCellulesX(*(listeDesConstructions+i),
                                      l, h, e))
            {
                *(*(listeDesConstructions+i))=-1;
            }
        }
    }
}

void Traitement::filtreY(int **listeDesConstructions,
                         int l,int h,int e)
{
    for(int i=0;
        i<1000000 && !estFinDeListe(*(listeDesConstructions+i));
        i++)
    {
        if(*(*(listeDesConstructions+i))==0)
        {
            if(!alignementDeCellulesY(*(listeDesConstructions+i),
                                      l, h, e))
            {
                *(*(listeDesConstructions+i))=-1;
```

```
            }
        }
    }
}

void Traitement::filtreZ(int **listeDesConstructions,
                         int l,int h,int e)
{
    for(int i=0;
        i<1000000 && !estFinDeListe(*(listeDesConstructions+i));
        i++)
    {
        if(*(*(listeDesConstructions+i))==0)
        {
            if(!alignementDeCellulesZ(*(listeDesConstructions+i),
                                      l, h, e))
            {
                *(*(listeDesConstructions+i))=-1;
            }
        }
    }
}

bool Traitement::alignementDeCellulesX(int *tabBin,int l,int h,int e)
{
    for(int z=0; z<h; z++)
    {
        for(int y=0; y<e; y++)
        {
```

```
            int x=0;
            while(x<l && *(tabBin + 1 + x + l*y + l*e*z)!=1)
            {
                x++;
            }
            int changements=0;
            while(x<l-1)
            {
                if(*(tabBin + 1 + x + l*y + l*e*z)!=
                                    *(tabBin + 1 + (x+1) + l*y + l*e*z))
                {
                    changements++;
                }
                x++;
            }

            if(changements > 1)
            {
                return false;
            }
        }
    }

    return true;
}

bool Traitement::alignementDeCellulesY(int *tabBin,int l,int h,int e)
{
    for(int z=0; z<h; z++)
```

```
    {
      for(int x=0; x<l; x++)
      {
          int y=0;
          while(y<e && *(tabBin + 1 + x + l*y + l*e*z)!=1)
          {
              y++;
          }
          int changements=0;
          while(y<e-1)
          {
              if(*(tabBin + 1 + x + l*y + l*e*z)!=
                                  *(tabBin + 1 + x + l*(y+1) + l*e*z))
              {
                  changements++;
              }
              y++;
          }

          if(changements > 1)
          {
              return false;
          }
      }
    }

    return true;
}
```

```
bool Traitement::alignementDeCellulesZ(int *tabBin,int l,int h,int e)
{
    for(int y=0; y<e; y++)
    {
        for(int x=0; x<l; x++)
        {
            int z=0;
            while(z<h && *(tabBin + 1 + x + l*y + l*e*z)!=1)
            {
                z++;
            }
            int changements=0;
            while(z<h-1)
            {
                if(*(tabBin + 1 + x + l*y + l*e*z)!=
                                    *(tabBin + 1 + x + l*y + l*e*(z+1)))
                {
                    changements++;
                }
                z++;
            }

            if(changements > 1)
            {
                return false;
            }
        }
    }
```

```
    return true;
}

bool Traitement::uneCelluleParPlaqueXY(int *tabBin,
                                       int l, int h, int e)
{
    for(int z=0; z<h; z++)
    {
        int cubesParPlaque=0;
        for(int y=0; y<e; y++)
        {
            for(int x=0; x<l; x++)
            {
                if(*(tabBin + 1 + x + l*y + l*e*z)==1)
                {
                    cubesParPlaque++;
                }
            }
        }
        if(cubesParPlaque==0)
        {
            return false;
        }
    }

    return true;
}

bool Traitement::uneCelluleParPlaqueXZ(int *tabBin,
```

```
                                        int l, int h, int e)
{
    for(int y=0; y<e; y++)
    {
        int cubesParPlaque=0;
        for(int z=0; z<h; z++)
        {
            for(int x=0; x<l; x++)
            {
                if(*(tabBin + 1 + x + l*y + l*e*z)==1)
                {
                    cubesParPlaque++;
                }
            }
        }
        if(cubesParPlaque==0)
        {
            return false;
        }
    }

    return true;
}

bool Traitement::uneCelluleParPlaqueYZ(int *tabBin,
                                        int l, int h, int e)
{
    for(int x=0; x<l; x++)
    {
```

```
        int cubesParPlaque=0;
        for(int y=0; y<e; y++)
        {
            for(int z=0; z<h; z++)
            {
                if(*(tabBin + 1 + x + l*y + l*e*z)==1)
                {
                    cubesParPlaque++;
                }
            }
        }
        if(cubesParPlaque==0)
        {
            return false;
        }
    }

    return true;
}

////////////////////////////////////////////////////////////////////////
// CONNEXITÉ
////////////////////////////////////////////////////////////////////////


void Traitement::filtreConnexe(int **listeDesConstructions, int l,
                               int h, int e)
{
    int taillePrisme = l*h*e+2*l*h+2*l*e+2*e*h+ 4*(l+2)+4*h+4*e;
```

```
    for(int i=0;
        i<1000000 && !estFinDeListe(*(listeDesConstructions+i));
        i++)
    {
        if(*(*(listeDesConstructions+i))==0)
        {
            int *prismeEnveloppe=
                                ajoutCouche(*(listeDesConstructions+i),
                                            taillePrisme, l,h,e);

            if(!pointIsole(prismeEnveloppe, taillePrisme, l, h, e))
            {

                if(!estConnexe(prismeEnveloppe, taillePrisme, l, h, e))
                {
                    *(*(listeDesConstructions+i))=-1;
                }
            }
            else
            {
                *(*(listeDesConstructions+i))=-1;
            }

            delete[] prismeEnveloppe;
        }
    }
}
```

```
int *Traitement::ajoutCouche(int *tabBin, int taillePrisme, int l,
                             int h, int e)
{
    int *prismeEnveloppe=new int[taillePrisme + 1];

    int positionDansTabBin=1;
    for(int i=1; i<= (l+2)*(e+2); i++)
    {
        *(prismeEnveloppe + i)=-1;
    }

    for(int z=1; z<=h; z++)
    {
        for(int i=1; i<=l+2; i++)
        {
            *(prismeEnveloppe + z*(l+2)*(e+2) + i)=-1;
        }

        for(int y=1; y<=e; y++)
        {
            *(prismeEnveloppe + z*(l+2)*(e+2) + y*(l+2) + 1)=-1;

            for(int x=1; x<=l; x++)
            {
                *(prismeEnveloppe + z*(l+2)*(e+2) + y*(l+2) + 1 + x)=
                    *(tabBin + positionDansTabBin);
                positionDansTabBin++;
            }
```

```
            *(prismeEnveloppe + z*(l+2)*(e+2) + y*(l+2) +1+l+1)=-1;
        }

        for(int i=1; i<=l+2; i++)
        {
            *(prismeEnveloppe + (z+1)*(l+2)*(e+2) - (l+2) + i)=-1;
        }
    }

    for(int i=1; i<= (l+2)*(e+2); i++)
    {
        *(prismeEnveloppe + (h+1)*(l+2)*(e+2) + i)=-1;
    }

    return prismeEnveloppe;
}

bool Traitement::pointIsole(int *prismeEnveloppe, int taillePrisme,
                            int l, int h, int e)
{
    for(int i=1; i<=taillePrisme; i++)
    {
        if(*(prismeEnveloppe + i)>0)
        {
            if( *(prismeEnveloppe + i - 1)<=0 &&
                *(prismeEnveloppe + i + 1)<=0 &&
                *(prismeEnveloppe + i - (l+2))<=0 &&
                *(prismeEnveloppe + i + (l+2))<=0 &&
                *(prismeEnveloppe + i - (l+2)*(e+2))<=0 &&
```

```
                *(prismeEnveloppe + i + (l+2)*(e+2))<=0 )
            {
                return true;
            }
        }
    }

    return false;
}

bool Traitement::estConnexe(int *prismeEnveloppe, int taillePrisme,
                            int l, int h, int e)
{
    numeroter(prismeEnveloppe, taillePrisme);

    bool changement = true;
    while(changement == true)
    {
        changement = false;

        for(int i=1; i<=taillePrisme; i++)
        {
            if(*(prismeEnveloppe + i)>0)
            {
                if(*(prismeEnveloppe + i - 1) > *(prismeEnveloppe + i))
                {

                    changement = true;
                    *(prismeEnveloppe + i - 1)=*(prismeEnveloppe + i);
```

```
}

if(*(prismeEnveloppe + i + 1) > *(prismeEnveloppe + i))
{
    changement = true;
    *(prismeEnveloppe + i + 1)=*(prismeEnveloppe + i);
}

if(*(prismeEnveloppe+i-(l+2)) > *(prismeEnveloppe+i))
{
    changement = true;
    *(prismeEnveloppe+i-(l+2)) = *(prismeEnveloppe+i);
}

if(*(prismeEnveloppe+i+(l+2)) > *(prismeEnveloppe+i))
{
    changement = true;
    *(prismeEnveloppe+i+(l+2)) = *(prismeEnveloppe+i);
}

if(*(prismeEnveloppe+i-(l+2)*(e+2)) >
                                    *(prismeEnveloppe+i))
{
    changement = true;
    *(prismeEnveloppe+i-(l+2)*(e+2)) =
                                    *(prismeEnveloppe+i);
}

if(*(prismeEnveloppe+i+(l+2)*(e+2)) >
```

```
                                                    *(prismeEnveloppe+i))
                {
                    changement = true;
                    *(prismeEnveloppe+i+(l+2)*(e+2)) =
                                                    *(prismeEnveloppe+i);
                }
            }
        }
    }

    int i = 1;
    while(*(prismeEnveloppe + i)<=0)
    {
        i++;
    }
    int marque = *(prismeEnveloppe + i);

    for(int j = 1; j<=taillePrisme; j++)
    {
        if(*(prismeEnveloppe + j)>0 && *(prismeEnveloppe + j)!=marque)
        {
            return false;
        }
    }

    return true;
}

void Traitement::numeroter(int *prismeEnveloppe, int taillePrisme)
```

```
{
    int numero = 1;
    for(int i=1; i<=taillePrisme; i++)
    {
        if(*(prismeEnveloppe + i)==1)
        {
            *(prismeEnveloppe + i)=numero;
            numero++;
        }
    }
}

////////////////////////////////////////////////////////////////////////
//Les procédures qui suivent sont secondaires
////////////////////////////////////////////////////////////////////////

void Traitement::afficherConstruction(int *constructionBin, int taille)
{
    std::cout<<"\n";
    for(int i=1; i<=taille; i++)
    {
        std::cout<<*(constructionBin+i);
    }
}

int *Traitement::copierTabBin(int *premierDuPaqSuivant_nbrPoly,
                              int volPrisme)
{
    int *premierDuPaquet=new int[volPrisme+2];
```

```
    for(int i=0; i<volPrisme +2; i++)
    {
        *(premierDuPaquet+i)=*(premierDuPaqSuivant_nbrPoly+i);
    }

    return premierDuPaquet;

}

//************************** Fin du fichier ************************//
```

BIBLIOGRAPHIE

[1] Barcucci, E., Frosini, A., et Rinaldi S.(2005). On directed-convex polyominoes in a rectangle, *Discrete Mathematics*, 298, 62-78.

[2] Bousquet-Mélou, M.(1994). Codage des polyominos convexes et équations pour l'énumération suivant l'aire, *Discrete Applied Mathematics*, 48, 21-43.

[3] Brlek, S., Labelle, G. et Lacasse, A.(2005). Algorithms for polyominoes based on the discrete Green theorem, *Discrete Applied Mathematics*, 147, 187-205.

[4] Deutsch, E.(2004). Enumerating symmetric directed convex polyominoes, *Discrete Mathematics*, 280, 225-231.

[5] Dubernard, J.P. et Dutour, I.(1996). Enumération de polyominos convexes dirigés, *Discrete Mathematics*, 157, 79-90.

[6] Golomb, S. *Polyominoes*, Princeton : Princeton University Press, **1994.**

[7] Goupil, A., Cloutier, H. Nouboud, F.(2010). Enumeration of inscribed polyominos in a rectangle, *Discrete Applied Mathematics*, 158, 2014-2023

[8] Goupil, A., Cloutier, H. (2011). 3D polyominoes inscribed in a rectangular prism, *Formal Power Series and Algebraic Combinatorics* (FPSAC 2011)

[9] Jensen, I.(2008). Enumerations of lattice animals and trees, *ARXIV.*

[10] Rassart, E.(1999). *Énumération de certaines classes de polyominos et de chemins autoévitants*, mémoire, UQAM, 58-69.

[11] Robitaille, A.(1997) *Dénombrement des classes de symétries des polyominos convexes*, mémoire, UQAM, 8-13.

[12] Xin, G.(2010) Communications personnelles.