UNIVERSITÉ DU QUÉBEC À MONTRÉAL

GÉNÉRATION EXHAUSTIVE DE POLYOMINOS

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

JÉRÔME TREMBLAY

MAI 2016

Remerciements

Merci à mon directeur, Srečko, de m'avoir inlassablement poussé vers la réussite. Merci à mes co-auteurs, Alain, Hugo, Jérôme, Jonathan et Romaine, d'avoir partagé mon aventure dans le merveilleux monde de la recherche fondamentale. Merci à mon ange gardien, Johanne, de m'avoir écouté, réglé mes problèmes et de façon générale pensé à tout ! Merci à mes collègues, Alexandre, Alexandra, Anne, Christian, Christophe, Christophe, Franco, François, Gilbert, Gisèle, Manon, Steven, Steven, Sylvie et Vestislav, j'ai apprécié chaque jour de travail grâce à vous tous. Merci à ma conscience, Marie-Claude, de m'avoir permis de survivre à la bureaucratie. Merci à mes amis, Catherine, Didier, Hugues, Judith, Jean-Philippe, Malorie, Marco, Marie, Maxime, Nicolas, Nicolas, Sébastien et bien d'autres, d'avoir fait passer toutes ces années si rapidement. Merci à ma douce, Catherine, de m'avoir soutenu et encouragé tout ce temps. Merci à Pierre, de m'avoir ouvert les portes du LaCIM.

TABLE DES MATIÈRES

LISTE DES FIGURES v
LISTE DES TABLEAUX vii
RÉSUMÉ ix
INTRODUCTION 1
CHAPITRE I
PRÉLIMINAIRES 5
1.1 Les polyominos 5
1.2 Représentations 8
CHAPITRE II
ENVELOPPE EXTERNE D'UN CHEMIN DISCRET 13
2.1 Les chemins 14
2.2 Enveloppes externes et convexes 17
2.3 Algorithme 19
CHAPITRE III
GÉNÉRATION EXAUSTIVE 39
3.1 Les gominos 39
3.2 Un jeu bien solitaire 40
3.2.1 Position de départ et zones interdites 41
3.2.2 Règles de déplacement 42
3.2.3 Fin de partie 44
3.3 L'algorithme de génération 46
3.3.1 4-trous 47
3.3.2 8-trous 48
3.3.3 Optimisations 49
3.3.4 Résultats expérimentaux 53
CONCLUSION 55
BIBLIOGRAPHIE 57

LISTE DES FIGURES

Figure		Page
0.1	Quelques polyominos 4-connexes bien connus	3
1.1	Un chemin	5
1.2	Exemples de polyominos	6
1.3	Les contours	7
1.4	Polyominos et trous	7
1.5	Un polyomino et son graphe associé	8
1.6	Un polyomino et son animal	9
1.7	Disques de rayon 1.5 et leur équivalent discret.	9
1.8	Directions associées à deux alphabets de Freeman	10
1.9	Un polyomino, son contour et son mot de contour	11
2.1	L'algorithme glouton de la chenille produit un chemin simple.	13
2.2	Deux codages du même chemin	15
2.3	Boîte englobante et factorisation standard	16
2.4	(a) Le graphe G_P codé par le mot $w =$ `001233`; (b) Le plongement anti-horaire de $\mathcal{G}(P)$ dans $\mathbb{R}^2$; (c) Le schéma de rotation de $\mathcal{G}(P)$.	17
2.5	Chemin `021` et son enveloppe externe	18
2.6	Orientation du parcours de $\mathcal{G}(w)$	20
2.7	Arbre radix et sa représentation binaire.	22
2.8	Les 4 premiers niveaux de l'arbre radix de $\mathbb{N}^2$.	23
2.9	Cinq premiers niveaux de l'arbre radix de $\mathbb{N}^2$ initialisé en $(15, 4)$.	25
2.10	Changement de quadrant	32
2.11	Les points du plan regroupés par parent	34

3.1 Un polyomino simplement 4-connexe et son gomino. 40

3.2 Motifs en X 41

3.3 Factorisation NSEW du contour 41

3.5 L'anatomie d'un coup 42

3.4 Position de départ et zones interdites 42

3.6 Coups permis (à rotation près) 43

3.7 Ordre du 8-voisinage 45

3.8 Un gomino sans libertés pour **W** 48

3.9 Les quatres positions initiales pour énumérer les trous de 3.9a 49

3.11 Nombre minimal de pierres noires requises 50

3.11 Nombre minimal de pierres noires requises 51

3.12 Temps de calcul des deux premières colonnes du tableau 3.1. (L'algorithme original est marqué par des ∘ tandis que l'algorithme optimisé est marqué par des ×) 54

LISTE DES TABLEAUX

Tableau Page

2.1 Voisinage des parents lorsque a et b sont voisins. 24

2.2 Récapitulation des méthodes de détection d'intersection 36

3.1 Nombre de polyominos de périmètre de site n selon leur connectivité. . . 53

RÉSUMÉ

La géométrie digitale est un domaine d'études développé récemment, à cause du développement des outils numériques. Ce mémoire examine deux problèmes importants dans le domaine de la géométrie digitale, à savoir le calcul de l'enveloppe extérieure d'un chemin discret et également la génération exhaustive de polyominos. Les deux reposent sur le codage des objets dans le plan discret, assimilé à la grille carrée $\mathbb{Z} \times \mathbb{Z}$, par des mots sur l'alphabet de Freeman $\mathcal{F} = \{\mathsf{0}, \mathsf{1}, \mathsf{2}, \mathsf{3}\}$, correspondant aux déplacements élémentaires sur la grille carrée.

Pour le problème de l'enveloppe extérieure d'un chemin quelconque (se recoupant ou pas), on utilise une structure de données utilisant deux arbres quaternaires superposés, introduite par Brlek, Provençal et Koskas. Utilisant un parcours mimant l'algorithme de la main droite dans un labyrinthe on obtient un algorithme linéaire en temps et en espace pour effectuer le calcul de l'enveloppe extérieure.

Dans le cas de la génération de polyominos codés par un chemin décrivant leurs périmètres de site, on utilise une adaptation du jeu de go pour construire à partir de l'origine et dans le sens antihoraire le chemin qui les borne. L'algorithme est exhaustif, et génère un polyomino chaque fois que le joueur noir gagne.

Mots-clés : polyominos, gominos, périmètre de site, chemins, enveloppe externe, génération exhaustive, algorithme, mots, arbre radix.

INTRODUCTION

L'avènement des ordinateurs au cours des dernières décennies a apporté des changements technologiques qui ont bouleversé les communications, la transmission d'informations et le traitement des données numériques.

En particulier, l'acquisition, la synthèse et le traitement des images constituent aujourd'hui un champ d'activité fertile pour le développement de nombreuses applications qu'on voit dans la vie de tous les jours. Astronomie, médecine, météorologie, surveillance, vision, sécurité et même le transport sont autant de domaines qui bénéficient de ces avancées technologiques.

Ces technologies sont basées sur des structures de données et algorithmes opérant sur une matrice de points (pixels) étudiés depuis longtemps et le corpus des connaissances acquises forme le coeur du domaine de la géométrie digitale. On s'accorde généralement pour situer la naissance de ce domaine en 1969 (Rosenfeld, 1969), bien qu'on trouve des algorithmes pour tracer des segments de droite bien avant (Bresenham, 1965) ainsi que d'autres notions qui se sont avérées équivalentes (Christoffel, 1875).

Le plan de pixels s'identifie naturellement au plan discret constitué par le produit cartésien $\mathbb{Z} \times \mathbb{Z}$, et une figure discrète dans le plan n'est rien d'autre qu'un sous-ensemble de $\mathbb{Z} \times \mathbb{Z}$. Les figures connexes dans le plan discret sont commodément représentées par les chemins qui les bornent. Ainsi le bord est constitué par un chemin, ou plusieurs chemins si la figure possède des trous.

Le polyomino est une de ces figures discrètes. Popularisé par S. Golomb (Golomb, 1954) comme jeu mathématique, le polyomino est un objet combinatoire rencontré dans plusieurs problèmes intéressants tels les pavages (Beauquier et Nivat, 1991; Golomb, 1996) ainsi que certains jeux (Gardner, 1958) et problèmes d'énumération (Jensen, 2001; Jen-

sen et Guttmann, 2000). De plus, certaines classes de polyominos codent efficacement les partages d'entiers et sont utilisés en théorie des nombres et en physique mathématique pour représenter le groupe symétrique (Fulton, 1997). Une représentation commode de tout polyomino est donc réalisée par les chemins qui le bornent.

Ce mémoire concerne deux problèmes importants dans le domaine de la géométrie digitale, à savoir le calcul de l'enveloppe extérieure d'un chemin discret et également la génération exhaustive de polyominos.

Les objets convexes tiennent un rôle important dans plusieurs branches des mathématiques, comme l'analyse fonctionnelle, l'optimisation, les probabilité et la physique mathématique (voir (Gruber, 2007) pour un traitement détaillé de la géométrie convexe et ses applications.) En géométrie euclidienne, étant donné un ensemble fini de points, le problème de trouver le plus petit ensemble convexe les contenant tous a mené à l'introduction de la notion géométrique d'enveloppe convexe. Le calcul de l'enveloppe convexe s'est révélé être l'un des algorithme fondamental de la géométrie computationelle avec des applications allant de la recherche opérationelle (Sherali et Adams, 1994) à l'automatisation du design (Kim, 1992). Il est également largement utilisé en graphisme, en particulier dans le traitement d'images (Kim *et al.*, 1997). Il est bien connu que dans le cas euclidien, les algorithmes de calcul de l'enveloppe convexe d'un ensemble $S \subset \mathbb{R}^2$ s'exécutent en temps $\mathcal{O}(n \log n)$ où $n = |S|$ (voir (Graham, 1972; Chan, 1996)). On peut même montrer que de tels algorithmes sont optimaux à une constante linéaire près (Avis, 1982; van Emde Boas, 1980).

Néanmoins, en restreignant le problème au calcul de l'enveloppe convexe de polygones simples, une borne asymptotique linéaire est établie (McCallum et Avis, 1979; Melkman, 1987). La version digitale de ce problème est un peu plus compliquée. On peut par exemple calculer l'enveloppe convexe d'un ensemble de pixels S en commençant par calculer l'enveloppe convexe Euclidienne de S et ensuite en digitalisant le résultat (Chaudhuri et Rosenfeld, 1998). Ceci nous donne automatiquement un pire cas en $\mathcal{O}(n \log n)$. Dans le cas discret, la situation est surprenamment plus simple grâce à l'aide

de la combinatoire des mots, qui a récemment mené au développement d'outils efficaces pour l'étude de la géométrie digitale (Provençal, 2008; Blondin Massé, 2012). Par exemple, des bornes asymptotiques linéaires ont été établues lorsqu'on considère un chemin discret codé par une suite de pas élémentaires. En effet, Brlek et al. ont développé un algorithme linéaire pour calculer l'enveloppe convexe discrète d'un chemin fermé auto-évitant sur la grille carré (Brlek *et al.*, 2009). Il est basé sur un algorithme de Duval (Duval, 1983) linéaire en temps et en espace optimal pour factoriser un mot de Lyndon. La situation est plus compliquée dans le cas d'un chemin non-simple.

Au chapitre 2 nous présentons un algorithme linéaire en temps et en mémoire pour calculer l'enveloppe extérieure d'un chemin discret décrit par un mot sur le code de Freeman. L'utilisation d'une structure d'arbre radix augmenté (Brlek *et al.*, 2011) permet une représentation linéaire des points visités ainsi que leur liens de voisinage dans le plan discret $\mathbb{Z} \times \mathbb{Z}$. Construire cette structure de données est une opération linéaire en temps et en espace et combinée à l'algorithme de la main droite, bien connu pour la résolution de labyrinthe, permet d'obtenir un algorithme linéaire en temps et en espace pour calculer l'enveloppe externe. Ce résultat fut exposé à la conférence Internationale «18th IAPR International Conference, DGCI 2014, Siena, Italy, September 10-12, 2014.»

> Brlek, S., Tremblay, H., Tremblay, J. et Weber, R. (2014). Efficient computation of the outer hull of a discrete path. Dans Discrete Geometry for Computer Imagery - Proceedings, volume 8668 de Lecture Notes in Computer Science.

Il existe plusieurs types de polyominos, ceux que nous utiliserons ici sont des ensembles finis de carrés unités connexes par les arêtes dans le plan discret.

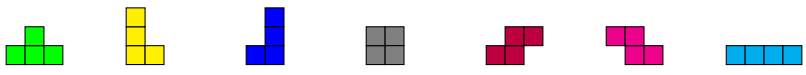


Figure 0.1: Quelques polyominos 4-connexes bien connus

L'énumération des polyominos en général est un problème difficile encore ouvert. Malgré cela, plusieurs sous-classes ont été énumérées avec succès en imposant différentes contraintes géométriques aux polyominos. Par exemple, il est bien connu (Pólya, 1969; Stanley, 1999) que le nombre de polyominos parallélogrammes, dont les lignes et les colonnes se croisant sur la diagonale principale sont contigües, de semi-perimètre $(n + 1)$

est le n-ième *nombre de Catalan* (séquence M1459 sur (Sloane, 2011)).

En revanche, le nombre a_n de polyominos simplement 4-connexe sans trous à n cellules a été calculé jusqu'à $n = 56$ (Jensen, 2001; Sloane, 2005) et le comportement asymptotique du nombre ces polyominos $\{a_n\}_{n\geq 0}$ est borné par la relation $\lim_{n\to\infty} \{a_n\}^{1/n} = \mu$ où $3.90 < \mu < 4.64$ (Jensen et Guttmann, 2000; Klarner et Rivest, 1972).

Le lecteur qui désire en savoir plus peut consulter Viennot (Viennot, 1992) pour l'énumération exacte de plusieurs classes de polyomino, ainsi que (Bousquet-Mélou, 1994; Bousquet-Mélou, 1996; Bousquet-Mélou, 1998) pour plus de résultats, et (Brlek *et al.*, 2006) pour l'énumération d'une sous-classe de tuiles parallélogrammes. Plus récemment, l'énumération de certaines familles de polyominos inscrits dans un rectangle a été obtenue dans (Goupil *et al.*, 2010).

Malgré le fait que ces problèmes aient été largement étudiés, plusieurs questions demeurent ouvertes. Par exemple, on ne connaît pas de formule close pour le nombre de polyominos, et la génération presque exhaustive par ordinateur (en utilisant le lemme de Burnside pour prendre en compte certaines symétries) demeure la seule façon de les compter. Plusieurs algorithmes existent tels ceux de Redelmeier (Redelmeier, 1981), un algorithme inductif, et celui Jensen (Jensen, 2001) plus rapide et basé sur les matrices de transfert (Stanley, 1999). Ces deux algorithmes sont exponentiels, et celui de Jensen gagne en vitesse au prix d'une consommation exponentielle de mémoire.

Au chapitre 3, nous exposerons en détail une autre méthode de génération exhaustive, introduite dans (Fortier *et al.*, 2013), basée sur la contrainte des polyominos selon leur *périmètre de site* (Bousquet-Mélou et Rechnitzer, 2003; Sieben, 2008). Le périmètre de site est le nombre de cellules extérieures à un polyomino qui touchent sa bordure. Cet algorithme s'inspire du jeu de go [1] et utilise deux types de pierres, noires et blanches. Le résultat présenté ici a fait l'objet de la publication suivante

> Fortier, J., Goupil, A., Lortie, J. et Tremblay, J. (2013). Exhaustive generation of gominoes. Theoretical Computer Science, 502, 76–87.

1. https ://fr.wikipedia.org/wiki/Go_(jeu)

CHAPITRE I

PRÉLIMINAIRES

Dans ce premier chapitre nous définissons les polyominos et nous introduisons quelques concepts de base s'y rattachants.

1.1 Les polyominos

Définition 1. *Une* cellule *est un carré unitaire* $[x, x+1] \times [y, y+1] \subseteq \mathbb{R}^2$. *Ses côtés sont appelés* arêtes.

Définition. *Deux cellules sont dites* 4-adjacentes *(resp.* 8-adjacentes*) si elles partagent au moins une arête (resp. un sommet).*

Une définition justifiée par le fait qu'une cellule possède un maximum de 4 ou 8 voisins selon le type de connexité.

Définition. *Un ensemble de cellules est* connexe *si toute paire de cellules est reliée par une suite de cellules adjacentes.*

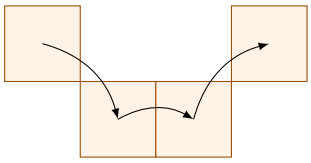
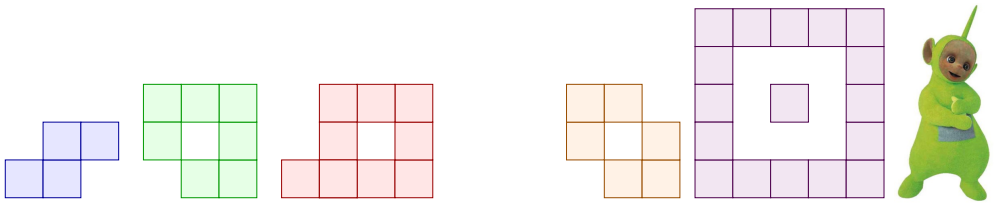


Figure 1.1: Un chemin dans un ensemble de cellules 8-connexe.



(a) Des polyominos (b) Pas des polyominos

Figure 1.2: Exemples de polyominos

Nous pouvons maintenant proposer une première définition du polyomino :

Définition 2. *Un* polyomino *est un ensemble de cellules* 4*-connexe.*

On remarque à la figure 1.2 que cette définition implique l'existence de polyominos contenant des "trous".

Définition 3. *Soit P un polyomino. Le* contour extérieur *de P, $C_e(P)$, est l'ensemble des cellules* 4*-connexe à P n'appartenant pas à P.*

Définition 4. *Soit P un polyomino. Son* contour intérieur *$C_i(P)$ est l'intersection entre P et les cellules* 4*-connexe à $C_e(P)$.*

Le contour extérieur est connu dans la littérature sous le nom *périmètre de site*. Pour générer un polyomino, nous générerons son contour défini comme suit (Bousquet-Mélou et Rechnitzer, 2003; Sieben, 2008) :

Définition 5. *Le* contour extérieur *d'un polyomino P est l'ensemble $P^+ \subseteq \mathbb{Z}^2 \backslash P$ de tous les points $b \in P^+$ qui sont* 4*-connexe à au moins un point de P. De la même manière, le* contour intérieur *de P est le sous-ensemble $P^- \subseteq P$ de points qui sont* 8*-connexes à un point de $\mathbb{Z}^2 \setminus P$.*

La figure 1.3 illustre le contour intérieur et le contour extérieur d'un polyomino. Il est clair que pour un polyomino donné, ses contours intérieur et extérieur sont uniques.

Quelques propriétés intéressantes :

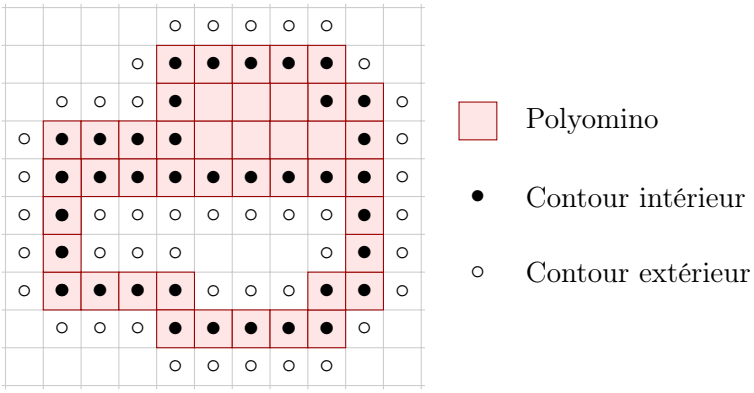


Figure 1.3: Les contours

- $C_e(P) \cap P = \emptyset$
- $C_i(P) \subseteq P$
- Pour tout $c \in C_e(P)$, il existe un $t \in P$ tel que c et t sont adjacentes par les côtés.

Trous et connexité. On obtient différentes sous-classes de polyominos en ajoutant des contraintes de connectivité. Un polyomino P est *simplement* 4-connexe (resp. *simplement* 8-connexe) si son complément $\mathbb{Z}^2 \setminus P$ est également 4-connexe (resp. 8-connexe). Par exemple, dans la figure 1.4, le polyomino P_1 est simplement 4-connexe, P_2 n'est pas simplement 4-connexe mais il est simplement 8-connexe et P_3 n'est pas simplement 8-connexe. L'algorithme décrit au chapitre 3 pour la génération exhaustive est assez

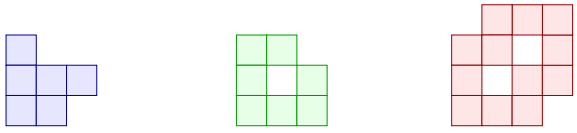


Figure 1.4: Polyominos et trous

général pour générer tous ces polyominos. Pour y parvenir, nous devons préciser la notion de *trou*.

Définition 6 (Trous)**.** *Soit P un polyomino. Un* 4-trou *(resp.* 8-trou*) de P est un sous-ensemble* 4*-connexe (resp.* 8*-connexe) maximal et fini de $\mathbb{Z}^2 \setminus P$.*

Ainsi, puisqu'un trou doit être maximal et fini, celui-ci doit être contenu dans un polyomino. Par exemple, dans la figure 1.4, P_2 a un 4-trou, qui dans ce cas est 8-connexe à une partie infinie de $\mathbb{Z}^2 \setminus P$. Ce n'est donc pas un 8-trou. Par contre, P_3 a un 8-trou, qui peut aussi être vu comme deux 4-trous.

1.2 Représentations

Nous avons déjà vu qu'un ensemble de cases connexes par les arêtes est un polyomino. Il y a plusieurs représentations naturelles qui s'imposent : graphe de la relation de connectivité, sa réalisation dans le plan discret $\mathbb{Z} \times \mathbb{Z}$, et le codage de son contour par des pas élémentaires.

Graphe de connexité. Parfois l'emplacement précis des cellules n'est pas important, c'est plutôt leur connectivité qui nous intéresse. On construit donc un graphe dont les sommets sont les cellules du polyomino et deux sommets sont reliés par une arête si et seulement si les cellules correspondantes sont 4-adjacentes.

Figure 1.5: Un polyomino et son graphe associé

Sous-ensemble de $\mathbb{Z}^2$. Le passage d'un ensemble de tuiles à un ensemble de points à coordonnées entières est naturel. Il suffit de choisir un point canonique sur la tuile, par exemple le plus en bas parmi les plus à gauche. Cette représentation est utilisée entre autre par les physiciens qui nomment ces objets des animaux (*lattice animals* en anglais) (Lubensky et Isaacson, 1979; Grosberg, 2014) qui surviennent dans des questions reliées à la percolation (Kraaikamp et Meester, 1995).

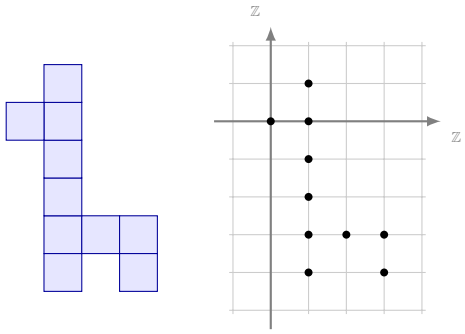
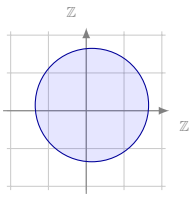


Figure 1.6: Un polyomino et son animal

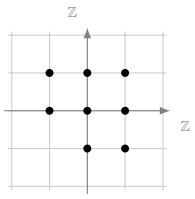
Bien que l'utilisation d'éléments de $\mathbb{Z}^2$ donne accès aux outils de la géométrie analytique, le passage du continue au discret reste délicat. Ainsi, même une définition aussi simple que le disque discret D de rayon r centré en C présente certains problèmes

$$D = \left\{ (x, y) \;\middle|\; (x - C_x)^2 + (y - C_y)^2 \leq r^2, x, y \in \mathbb{Z} \right\}.$$

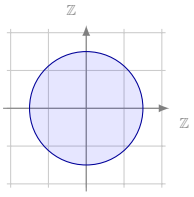
Dans (Brlek *et al.*, 2008) par exemple, on étudie ces disques et leur comportement lorsque C varie de façon continue dans $\mathbb{R}^2$ (figure 1.7).



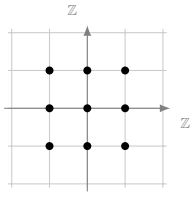
(a) Centré en $(0.15, 0.15)$



(b) Discrétisation de 1.7a



(c) Centré en $(0, 0)$



(d) Discrétisation de 1.7c

Figure 1.7: Disques de rayon 1.5 et leur équivalent discret.

Figure 1.8: Directions associées à deux alphabets de Freeman

Mot de contour. Un *alphabet* est un ensemble fini de symboles. Un *mot* m sur l'alphabet $\mathcal{A}$ est une séquence finie de symboles $a_i \in \mathcal{A}, i \in \mathbb{N}$.

$$m = a_1 a_1 a_2 \cdots a_n$$

On note $|m|$ la longueur du mot m, et m_i (ou $m[i]$ pour une notation plus près de l'informatique) sa $i^{\text{ième}}$ lettre. Le nombre d'occurrences de $\alpha \in \mathcal{A}$ dans m est noté $|m|_\alpha$ et le mot de longueur 0 est appelé le mot vide qu'on note ε. On remarque que les indices sont entre 1 et $|m|$ et non entre 0 et $|m| - 1$.

On note $\mathcal{A}^n$ l'ensemble des mots de longueur n, et $\mathcal{A}^*$ l'ensemble des mots finis.

Définition. *À l'aide de deux mots $a \in \mathcal{A}^m$ et $b \in \mathcal{A}^n$ on obtient par* concaténation*, ou* produit*, le mot $a \cdot b \in \mathcal{A}^{m+n}$, ou simplement ab.*

$$\begin{aligned} a &= a_1 a_2 \cdots a_m \\ b &= b_1 b_2 \cdots b_n \\ ab &= a_1 a_2 \cdots a_m b_1 b_2 \cdots b_n \end{aligned}$$

En 1961, Herbert Freeman propose de décrire des chemins en utilisant quatre "pas" élémentaires $(\rightarrow, \uparrow, \leftarrow, \downarrow) \simeq (0, 1, 2, 3)$ associés à des vecteurs tel qu'illustré à la figure 1.8 (Freeman, 1961). On note cet alphabet $\mathcal{F}$. On code un polyomino sans trou à l'aide de ces lettres en parcourant son bord, chaque lettre correspondant à un pas unitaire sur le chemin. On verra aux sections 3.3.1 et 3.3.2 comment étendre ce codage aux polyominos avec trou.

L'alphabet de virages décrit quant à lui les changements de direction durant le parcours. On peut passer d'un mot f sur l'alphabet de Freeman à un mot v sur l'alphabet de virages en soustrayant les valeurs consécutives :

$$v_i = f_{i+1} - f_i \mod 4 \quad 1 \leq i \leq |f|$$

Pour cette raison on le nomme aussi alphabet des premières différences. On interprète les valeurs $(0, 1, 2, 3) \simeq$ (tout droit, gauche, demi-tour, droite). En nous limitant aux mots de contour, qui ne contiennent pas de demi-tour, l'alphabet des premières différences est donc $\{0, 1, 3\}$.

Un 4-*chemin* (respectivement 8-*chemin*) est un chemin qui n'utilise que des pas unité horizontaux et verticaux (respectivement horizontaux, verticaux et diagonaux) . Si ce chemin est fermé et simple, c'est-à-dire que le point d'origine est visité exactement 2 fois, tandis que tous les autres points sont visités au maximum une fois, alors ce chemin décrit le contour d'un polyomino sans trou.

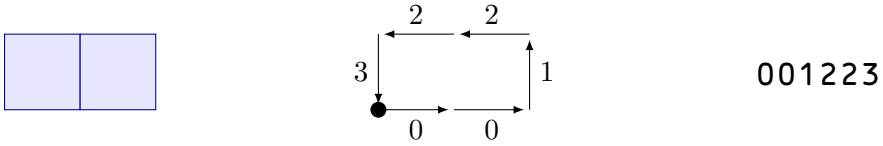


Figure 1.9: Un polyomino, son contour et son mot de contour

CHAPITRE II

ENVELOPPE EXTERNE D'UN CHEMIN DISCRET

Nous présenterons dans ce chapitre un algorithme linéaire pour calculer l'enveloppe extérieure d'un chemin discret décrit par un code de Freeman. Le lien étroit entre un contour et un chemin simple mène à l'utilisation d'une structure d'arbre radix augmenté introduite par (Brlek *et al.*, 2011) pour détecter les intersections, qui permet un stockage linéaire des points visités ainsi que leur liens de voisinage dans le plan discret $\mathbb{Z} \times \mathbb{Z}$. Cette structure de données combinée à l'algorithme de la main droite, bien connu pour la résolution de labyrinthe, permet d'obtenir un algorithme linéaire pour calculer l'enveloppe externe. Les résultats de ce chapitre ont été présentés à la conférence internationale DGCI 2014 (Brlek *et al.*, 2014).

Figure 2.1: L'algorithme glouton de la chenille produit un chemin simple.

2.1 Les chemins

Définition 7. *L'*ensemble de pas unitaires *est l'ensemble des points de* $\mathbb{Z}^2$ *accessibles à partir de* $(0,0)$ *en une seule opération.*

Les pas élémentaires de l'alphabet de Freeman forment un ensemble de pas unitaire.

$$\mathcal{F} = \{0,1,2,3\} \simeq \{(1,0),(0,1),(-1,0),(0,-1)\}$$

Définition 8. *Soit* P *un ensemble de pas unitaires. Un* chemin $c = c_0, c_1, ..., c_{n-1}$ *est une suite de points tels que pour tout* $i > 0$*, il existe au moins un* $p \in P$ *tel que* $c_i = c_{i-1} + p$*.*

On remarque que c_0 est indépendant de P. On peut ancrer un chemin n'importe où.

Proposition 1. *Soit* $P = \{(1,0),(0,1),(-1,0),(0,-1)\}$*. Pour tout chemin* C *à pas unitaires dans* P *enraciné en* $c_0 \in \mathbb{Z}^2$*, on a* $C \subset \mathbb{Z}^2$*.*

Définition 9. *Un chemin est* discret *lorsque tous ses points sonts dans* $\mathbb{Z}^2$*.*

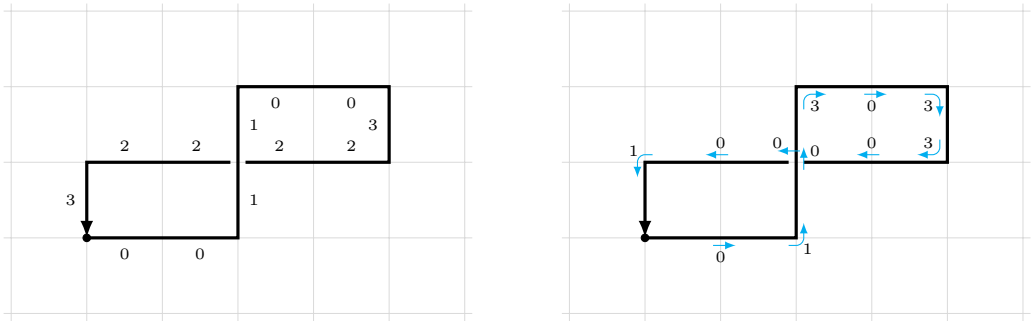
On vérifie facilement que si c_0 est à coordonnées entières, un chemin est discret.

Définition 10. *Un chemin est* fermé *s'il existe un pas unitaire* p *tel que* $c_0 = c_{n-1} + p$*.*

Définition 11. *Un chemin est* simple *si ses points sont visités exactement une fois. Un chemin simple peut être* fermé *si seul son point d'origine est visité une deuxième fois, au dernier pas.*

Il y a une bijection entre les cellules et $\mathbb{Z}^2$ obtenue en associant $(a,b) \in \mathbb{Z}^2$ au carré unitaire dont le coin inférieur gauche est de coordonnée (a,b). Ainsi, on peut considérer les cellules comme des éléments de $\mathbb{Z}^2$. Par définition, un *ensemble discret* S est un ensemble de cellules, i.e. $S \subset \mathbb{Z}^2$.

Une façon pratique de représenter un ensemble connexe discret sans trous est d'utiliser un mot décrivant son contour. Puisque le contour d'un ensemble discret est un chemin, on utilise le codage de Freeman qui représente efficacement un chemin discret dans $\mathbb{Z}^2$.



(a) $w = \mathsf{001100322223}$; (b) $\mathbf{\Delta}(w) = \mathsf{01030330001}$;

Figure 2.2: Deux codages du même chemin

Il est clair de ces définitions que tout chemin discret C est représenté par un mot $w \in \mathcal{F}^*$ où $\mathcal{F} = \{\mathsf{0}, \mathsf{1}, \mathsf{2}, \mathsf{3}\}$ est l'alphabet de Freeman. Il est intéressant de mentioner ici que dans le cas d'un chemin discret fermé, w est unique à permutation circulaire des lettres près. On peut vérifier à l'aide de la figure 2.2 que toute permutation circulaire du mot $w = \mathsf{001100322223}$ représente le même chemin, que w est non simple, fermé, et que

$$\mathbf{\Delta}(w) = \mathsf{01030330001}$$

code bien les virages de w.

Tout chemin est contenu dans un plus petit rectangle, appelé la *boîte englobante* (ou *bounding box* en anglais). On définit W comme étant le point le plus bas parmi les points les plus à gauche sur cette boîte tel qu'illustré à la figure 2.3. On peut facilement obtenir W en temps linéaire en lisant le mot une première fois en conservant les extremums.

On termine cette section avec un bref rappel de topologie des graphes (voir (Gross et Tucker, 1987) pour un traitement plus complet du sujet). Soit P un chemin discret (i.e. une suite de points entiers) codés par le mot w. L'image de P comme un sous-ensemble de $\mathbb{R}^2$ est notée G_P tandis que le graphe de l'image plongé dans le dans le plan $\mathbb{R}^2$ est notée $\mathcal{G}(P)$.

Le *plongement* d'un graphe G sur une surface S fait correspondre des points de S aux

(a) (b)

Figure 2.3: Boîte englobante et factorisation standard

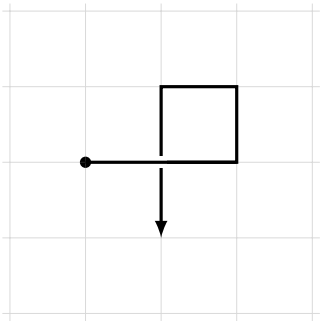
sommets de G et des *arcs simples* (images homéomorphiques de $[0, 1]$) aux arêtes de G de telle façon à ce que :

- Les extrémités de l'arc associé à l'arête e sont les points associés aux sommets aux extrémités de e.
- Aucun arc ne contient de points associés à un autre sommet.
- Deux arcs ne se croisent pas en un point contenu dans l'un des deux arcs.

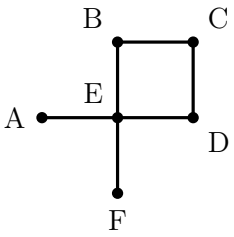
Intuitivement, cela signifie que les arcs ne se croisent pas et ne passent pas par un sommet.

Dans le cas du graphe associé au chemin discret P, un tel plongement est complètement déterminé en associant un ordre cyclique aux arêtes autour de chaque sommet de la façon suivante : d'abord on fixe une orientation en chaque point (par exemple anti-horaire). Ensuite, pour chaque sommet v dans G_P, on définit la permutation cyclique sur les arêtes entrantes de v. Ceci définit un schéma de rotation sur G_P. On peut montrer qu'un tel schéma de rotation est équivalent à un plongement orienté de G_P sur une surface. Par exemple, la figure 2.4 illustre le chemin P codé par $w = \texttt{001322}$, son graphe G_P et son plongement anti-horaire $\mathcal{G}_P$ dans $\mathbb{R}^2$.

Puisque par définition un plongement de P est planaire, la surface sur laquelle P est plongée se trouve partagée en un certain nombre de régions mutuellement exclusives



(a)



(b)

A	:	E
B	:	C E
C	:	B D
D	:	C E
E	:	A F D B
F	:	E

(c)

Figure 2.4: (a) Le graphe G_P codé par le mot $w = \texttt{001233}$; (b) Le plongement anti-horaire de $\mathcal{G}(P)$ dans $\mathbb{R}^2$; (c) Le schéma de rotation de $\mathcal{G}(P)$.

délimitées par $\mathcal{G}(P)$. On nomme ces régions *faces* et dans le cas d'une surface infinie comme $\mathbb{R}^2$, toutes les faces sont de taille finie sauf une. Cette face non-bornée est la *face externe* de $\mathcal{G}(P)$.

2.2 Enveloppes externes et convexes

On rappelle qu'en topologie, étant donné un ensemble S, le bord ∂S est l'ensemble des points dans l'adhérence de S qui ne sont pas à l'intérieur de S. L'*enveloppe externe* de S, notée Hull(S), est le bord de l'intersection de tous les ensembles discrets sans trous contenant S, i.e. le chemin simple suivant le contour de S. La définition 12 étend la notion d'enveloppe externe à tout chemin discret.

Définition 12. *Soit P un chemin discret. Alors, l'enveloppe externe de P, notée Hull(P) est la face externe de $\mathcal{G}(P)$, le plongement de P.*

On note que la définition 12 utilise le plongement de P plutôt qu'un ensemble discret pour décrire l'enveloppe externe. Ce choix permet de décrire l'enveloppe d'un chemin discret (i.e. un ensemble Euclidien d'aire 0). Par exemple, la figure 2.5 illustre l'enveloppe externe du chemin $w = 021$. On remarque aussi qu'en utilisant la définition 12, le bord d'un chemin est codé par un mot fermé. e.g. l'enveloppe externe du chemin codé par 0

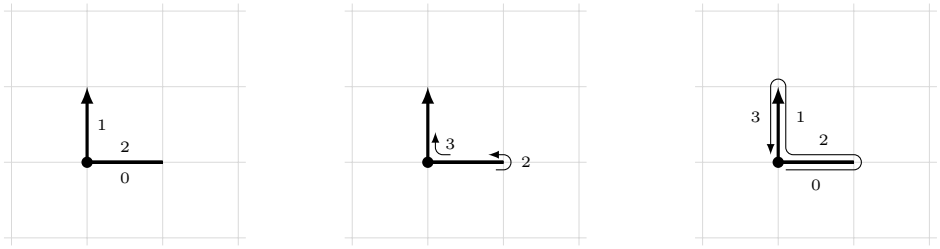


Figure 2.5: Chemin 021 et son enveloppe externe

est codé par 02. Ainsi, la définition 12 offre une généralisation adéquate de l'enveloppe externe aux chemins discrets quelconques. En effet, si P code le bord d'un ensemble discret sans trou S, alors P est simple et fermé par définition. Donc $P = \text{Hull}(P)$ et donc puisque $\text{Hull}(S)$ est le bord ∂S de S par définition, on a

$$\text{Hull}(S) = \partial S = P = \text{Hull}(P). \tag{2.1}$$

Puisqu'il y a une bijection entre les chemins discrets dans $\mathbb{Z}^2$ et les mots sur $\mathcal{F}$, on identifie P à son mot w et on écrit $\text{Hull}(w)$ plutôt que $\text{Hull}(P)$.

Pour finir, on rappelle certaines notions de base concernant la convexité digitale tout en référant le lecteur intéressé à (Brlek *et al.*, 2009; Provençal, 2008) pour une couverture plus complète du sujet. Soit S un ensemble discret 8-connexe. S est *digitalement convexe* s'il s'agit de la digitalisation de Gauss d'un sous-ensemble convexe R de $\mathbb{R}^2$, i.e. $S = \text{Conv}(R) \cap \mathbb{Z}^2$. L'*enveloppe convexe de* S, notée $\text{Conv}(S)$ est l'intersection de tous les ensembles convexes contenant S. Dans le cas d'un chemin simple w, $\text{Conv}(w)$ est donnée par la factorisation de Spitzer de w (voir (Spitzer, 1956; Brlek *et al.*, 2009)). Étant donné $w = w_1 w_2 \cdots w_n \in \{0,1\}^*$, on calcule le segment NW de la factorisation de la manière suivante : commencer avec la liste $(b_1, b_2, \ldots, b_n) = (w_1, w_2, \ldots, w_n)$. Si la pente $\rho(b_i) = |b_i|_1/|b_i|_0$ de b_i est strictement plus petite que celle de b_{i+1} pour un i, alors

$$(b_1, b_2, \ldots, b_k) = (b_1, \ldots, b_{i-1}, b_i b_{i+1}, b_{i+2}, \ldots, b_k).$$

En répétant le processus jusqu'à ce qu'il soit impossible de concaténer deux autres b_i, on

obtient la factorisation de Spitzer de w. Les parties NE, SE et SW de la factorisation sont obtenues par rotation.

2.3 Algorithme

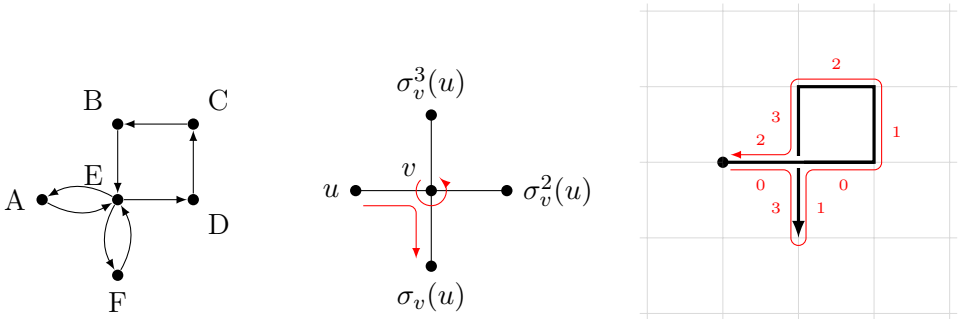
Soit $w \in \mathcal{F}^*$ un chemin discret et G_w sa représentation en graphe. Remarquons que l'application $g : w \to G_w$ n'est pas bijective puisqu'elle n'est pas injective (par exemple, $u = 0$ et $v = 02$ admettent le même graphe.). Nous avons vu à la section 2.1 que le plongement $\mathcal{G}(w)$ de G_w dans $\mathbb{R}^2$ donne lieu à un schéma de rotation si l'on fixe une orientation. Nous utilisons ce plongement pour calculer l'enveloppe externe de w (i.e. la face externe de $\mathcal{G}(w)$) : on fixe l'orientation $\mathcal{O}$ sur la surface $\mathbb{R}^2$ et on choisit le sommet le plus en bas parmis les plus à gauche, W, comme point de départ. Soit $e_0 = (W, v)$ un arc allant du sommet W vers le sommet v dans $\mathcal{G}(w)$. Par le choix de W, e_0 est une arête de la face externe de $\mathcal{G}(w)$. Par récurrence, soit $e_i = (u, v)$ un arc du sommet u vers le sommet v dans $\mathcal{G}(w)$ tel que e_i est une arête de la face externe du plongement $\mathcal{G}(w)$ et tel que e_i suis l'orientation fixée par $\mathcal{O}$. Ensuite, calculer $e_{i+1} = (v, \sigma_v(u))$ où σ_v est la permutation cyclique associée à v dans $\mathcal{G}(w)$. Alors e_{i+1} est aussi une arête de la face externe. En choisissant $\mathcal{O}$ anti-horaire, on peut répéter ce processus pour obtenir la face externe de $\mathcal{G}(w)$.

Par exemple, en utilisant le schéma de rotation défini pour $w = \texttt{001233}$ à la figure 2.4c et en commençant avec l'arc (A, E), on obtient la séquence d'arcs

$$(A, E), (E, F), (F, E), (E, D), (D, C), (C, B), (B, E), (E, A)$$

qui correspond à l'enveloppe externe de w (voir figure 2.6).

La justesse de la méthode découle de la « règle de la main droite » ou de « l'algorithme du mur » pour traverser un labyrinthe. En effet, étant donné l'arc (u, v), prendre l'arc adjacent $(v, \sigma_v(u))$ revient à « tourner à droite » au sommet v (voir figure 2.6b). Le principe sous-jacent étant donc de commencer sur un point de l'enveloppe externe et rester dessus lors du parcours en prenant systématiquement à droite à chaque intersection, en terminant lors du retour au point de départ. Ceci garantit que le parcours est



(a) (b) (c)

Figure 2.6: Orientation du parcours de $\mathcal{G}(w)$

précisément l'enveloppe externe de w.

Pour implémenter efficacement cette méthode, plusieurs problèmes doivent être résolus. Premièrement, tel qu'indiqué plus haut, le parcours doit commencer sur un point de l'enveloppe externe sinon le résultat pourrait ne pas décrire l'objet voulu. On choisit donc le point W associé au mot de contour w comme point de départ.

Deuxièmement, lorsqu'un parcours repasse par W on soit s'assurer que l'algorithme ne termine pas prématurément (le cas le plus simple étant codé par $w =$ `021`, voir la figure 2.5). Une solution simple consiste à construire une liste des voisins de W dans le chemin P, et de les retirer lorsqu'ils sont visités. Cette liste contient au plus deux éléments puisqu'aucun sommet de P ne peut se trouver à gauche ou en bas de W.

Le dernier problème à résoudre est la détection des intersections. Nous présenterons trois solutions solutions possibles avec des caractéristiques différentes. On cherche une fonction $f(x, y) \to \{1, 0\}$ qui nous indique si l'emplacement (x, y) a déjà été visité.

Première approche, l'adressage direct. Pour conserver la trace d'un chemin de longueur n dans $\mathbb{Z}^2$, on alloue immédiatement la mémoire pour l'ensemble de positions possibles dans un bloc contigu. On peut ainsi directement calculer l'emplacement de n'importe quelle position visitée. Un tableau $2n \times 2n$ contiendra toutes les positions atteignables par un chemin de longueur n enraciné en (n, n). Dans une telle structure de données,

l'accès à une position spécifique se fait en temps constant. Par contre les besoins en mémoire sont de l'ordre de $\mathcal{O}(n^2)$.

Deuxième approche, le tri. On ajoute chaque nouvelle position visitée à une liste triée. Plusieurs structures de données classiques se prêtent bien à cet usage comme les arbres rouge noir, les arbres B et les arbres AVL par exemple. Pour toutes ces structures de données bien connues, la consommation de mémoire est d'ordre $\mathcal{O}(n)$ et une opération de recherche ou d'insertion est d'ordre $\mathcal{O}(\log(n))$.

Troisième approche, l'arbre radix augmenté (Brlek *et al.*, 2011). Soit $G = (\mathcal{N}, E, V)$ un graphe où :

$\mathcal{N}$ Les sommets du graphe, on dira les noeuds, associés aux points de $\mathbb{Z}^2$.

E Les arêtes du graphe associées à la fonction ENFANT.

V Les arêtes du graphe associées à la fonction VOISIN.

On profite du fait que chaque accès est adjacent au précédent. Un noeud $N \in \mathcal{N}$ est défini par le quintuplet $N = (C, E, V, P, \Sigma)$ où $C \in \mathbb{Z}^2$ est la coordonnée cartésienne de ce noeud, $E = (E_0, E_1, E_2, E_3)$ où $E_i \in \mathcal{N}$ sont les quatre enfants, $V = (V_0, V_1, V_2, V_3)$ où $V_i \in \mathcal{N}$ sont les voisins, $P \in \mathcal{N}$ est le père et $\Sigma \in \mathbb{N}$ un compteur du nombre de fois où N est visité. De plus, on permettra d'étiqueter les V_i pour reconnaître un voisin *explicite*, spécifié par le chemin parcouru, ou *implicite*, ajouté par l'algorithme pour accélérer les calculs.

On notera le noeud du point p à l'aide d'une boîte $\boxed{p}$ et on utilisera la forme $\boxed{(x, y)}$ pour le noeud de coordonnée (x, y).

Tout d'abord on considère un arbre contenant les points du plan, construit d'après la fonction $e : \mathcal{N} \to \mathcal{N}^4$ qui permet de passer d'un noeud à ses 4 enfants :

$$e\left(\boxed{(x,y)}\right) = \left(\boxed{(2x,2y)}, \boxed{(2x+1,2y)}, \boxed{(2x+1,2y+1)}, \boxed{(2x,2y+1)}\right). \tag{2.2}$$

On définit la fonction ENFANT comme une restriction de e à l'enfant auquel on souhaite

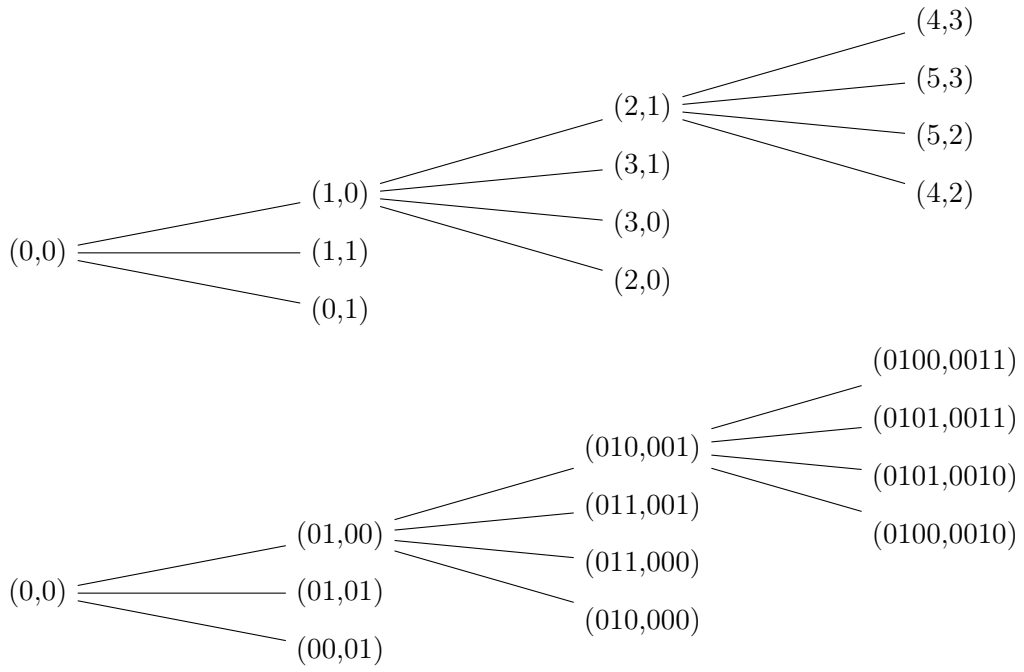


Figure 2.7: Arbre radix et sa représentation binaire.

accéder :

$$\text{ENFANT} : \mathcal{N} \times \mathbb{Z}^2 \to \mathcal{N}$$

où ENFANT $\left(\boxed{A}, (x, y)\right) = \boxed{(x, y)}$ est définie si et seulement si $\boxed{(x, y)}$ est bel et bien un enfant de $\boxed{A}$.

On remarque que les coordonnées des enfants sont obtenues en ajoutant 1 ou 0 à la représentation binaire des coordonnées du parent tel qu'illustré à la figure 2.7.

La fonction inverse

$$\text{PERE} : \mathcal{N} \to \mathcal{N}$$

permet de passer d'un noeud à son père. On peut aussi calculer directement les coordonnées du père :

$$f(x, y) = \left(\left\lfloor \frac{x}{2} \right\rfloor, \left\lfloor \frac{y}{2} \right\rfloor\right) \tag{2.3}$$

Le fait de connaître les coordonnées du père d'un point sans avoir de référence au noeud de ce point ou à celui de son père est crucial au fonctionnement de l'algorithme.

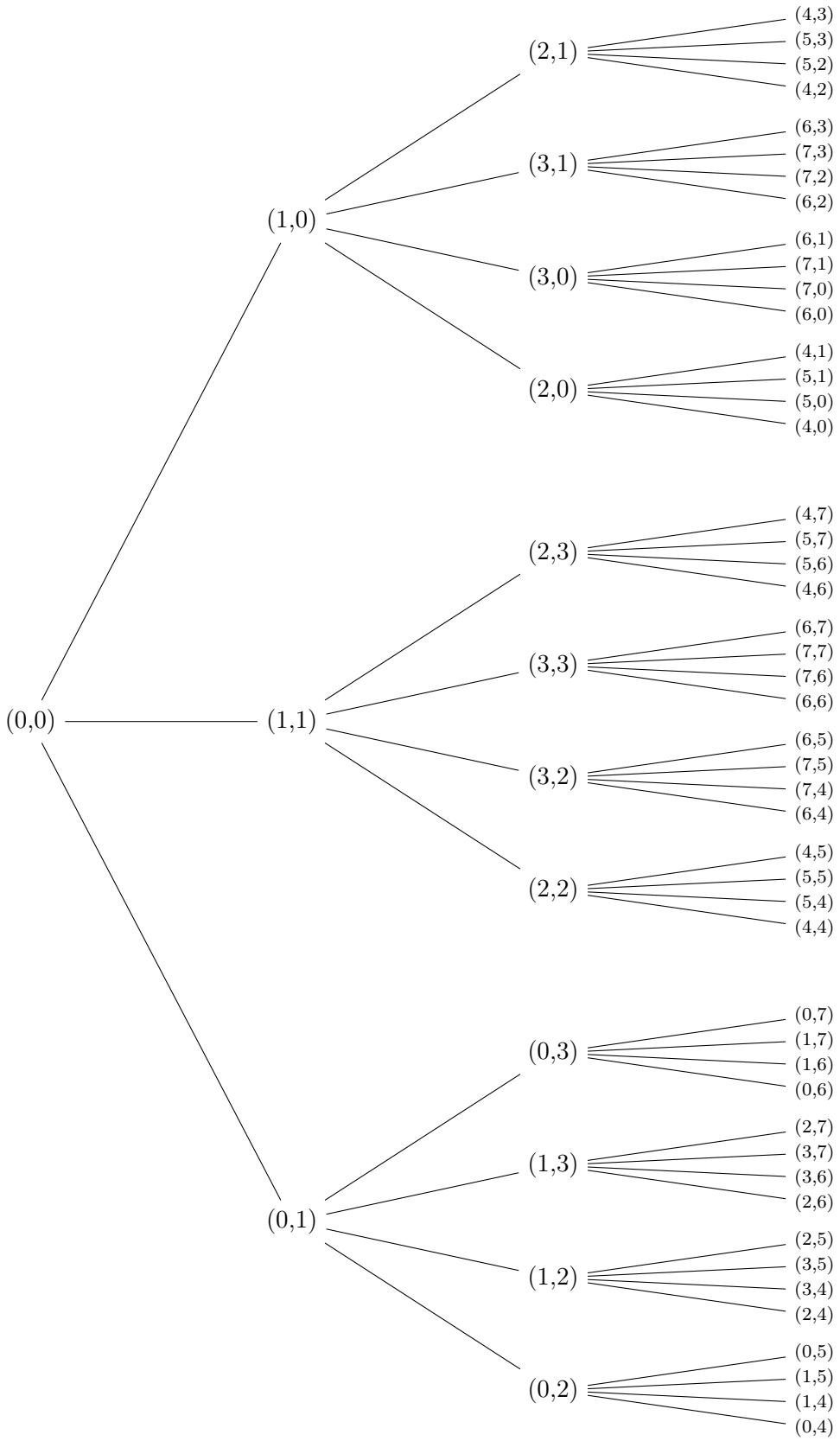


Figure 2.8: Les 4 premiers niveaux de l'arbre radix de $\mathbb{N}^2$.

a_x	Δ_x	b_x	$f(a)_x$	$f(b)_x$	$f(a)_x + \Delta_x$
$2x$	1	$2x+1$	x	x	$x+1$
$2x$	-1	$2x-1$	x	$x-1$	$x-1$
$2x+1$	1	$2x+1+1$	x	$x+1$	$x+1$
$2x+1$	-1	$2x+1-1$	x	x	$x-1$

Tableau 2.1: Voisinage des parents lorsque a et b sont voisins.

L'arbre est initialisé à une certaine position avec ses liens pères jusqu'à la racine en $\boxed{(0,0)}$. La racine est son propre père. La figure 2.9 illustre de quelle façon on divise le plan entre les différents niveaux de l'arbre.

La seule façon d'obtenir une référence à un nouveau noeud est par la fonction ENFANT, f ne fait que calculer les coordonnées.

Le lemme suivant stipule que des noeuds voisins ont le même père ou leurs pères sont voisins.

Lemme 1. *Soit a,b* $\in \mathbb{Z}^2$ *et* $\Delta \in \{(1,0),(0,1),(-1,0),(0,-1)\}$ *tel que* $a + \Delta = b$*, alors l'une des deux conditions suivantes est vraie :*

1. $f(a) = f(b)$
2. $f(a) + \Delta = f(b)$.

Démonstration. Puisque a et b sont voisins, on suppose sans perdre de généralité que $\Delta \in \{(-1,0),(1,0)\}$ donc $b_x = a_x + \Delta_x$ et $a_y = b_y$. On pose $x = f(a)_x$ alors par définition a_x est de forme $2x$ ou $2x+1$. Le tableau 2.1 récapitule les quatre cas possibles. On constate qu'on a bien $f(a)_x = f(b)_x$ ou $f(a)_x + \Delta_x = f(b)_x$ dans chacun des cas, tel que voulu.

□

L'algorithme VOISIN obtient une référence au voisin d'un noeud à l'aide d'un appel

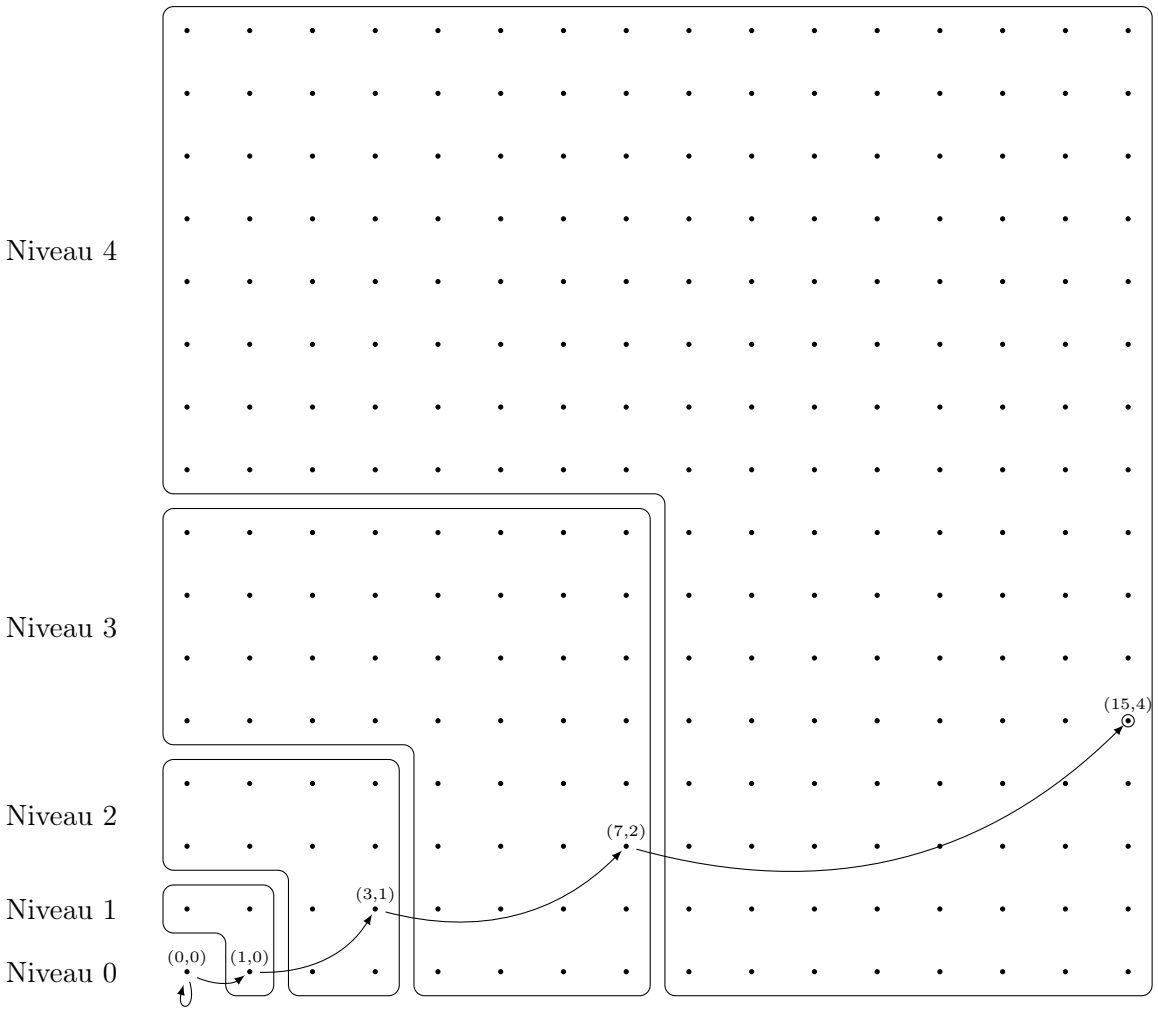


Figure 2.9: Cinq premiers niveaux de l'arbre radix de $\mathbb{N}^2$ initialisé en $(15, 4)$.

récursif à son père en vertu du lemme 1 :

$$\text{VOISIN} : \mathcal{N} \times \mathbb{Z}^2 \to \mathcal{N}.$$

Soit $\boxed{\text{a}} = (C, E, V, P, \Sigma) \in \mathcal{N}$ et $\boxed{\text{b}} \in \mathcal{N}$ des noeuds. On introduit la notation $\boxed{a} \overset{\Delta}{\to} \boxed{b}$ où $\Delta \in \mathcal{F}$, qui représente le voisin V_Δ du noeud $\boxed{a}$. Puisque notre structure de donnée est allouée dynamiquement, le voisin de $\boxed{a}$ en direction Δ n'est pas encore connu lors du premier appel VOISIN $\left(\boxed{a}, \Delta\right)$. On initialise $\boxed{a} \overset{\Delta}{\to} \boxed{b}$ avec $\boxed{a}_{V_\Delta} = \boxed{b}$.

Algorithme 1 VOISIN

Préconditions: $\boxed{o} \in \mathcal{N}$, $\Delta \in \mathcal{F}$

Postconditions: $\boxed{o} \overset{\Delta}{\to} \boxed{d}$ correctement initialisé

1: **function** VOISIN$\left(\boxed{o}, \Delta\right)$
2: **si** $\boxed{o} \overset{\Delta}{\to} \boxed{d}$ n'est pas initialisé **alors**
3: $p \leftarrow$ Pas élémentaire associé à Δ
4: $d \leftarrow o + p$
5: **si** d est un fils de o **alors**
6: $\boxed{d} \leftarrow$ ENFANT $\left(\boxed{o}, d\right)$
7: **sinon**
8: $\boxed{\text{pere}} \leftarrow$ PERE $\left(\boxed{o}\right)$
9: **si** $f(d) = pere$ **alors**
10: $\boxed{d} \leftarrow$ ENFANT $\left(\boxed{\text{pere}}, d\right)$
11: **sinon**
12: $\boxed{d} \leftarrow$ ENFANT $\left(\text{VOISIN}\left(\boxed{\text{pere}}, \Delta\right), d\right)$
13: initialiser $\boxed{o} \overset{\Delta}{\to} \boxed{d}$
14: initialiser $\boxed{d} \overset{\bar{\Delta}}{\to} \boxed{o}$
15: **retourner** $\boxed{d}$
16: **fin function**

On obtient ensuite naturellement l'algorithme PARCOURS en invoquant VOISIN avec chaque lettre du mot w.

$$\text{PARCOURS} : \mathcal{F}^* \times \mathcal{N} \to \mathcal{N}$$

La séquence qui suit illustre la progression de l'algorithme PARCOURS lors de la lecture du mot `222` à partir du point $(15, 4)$.

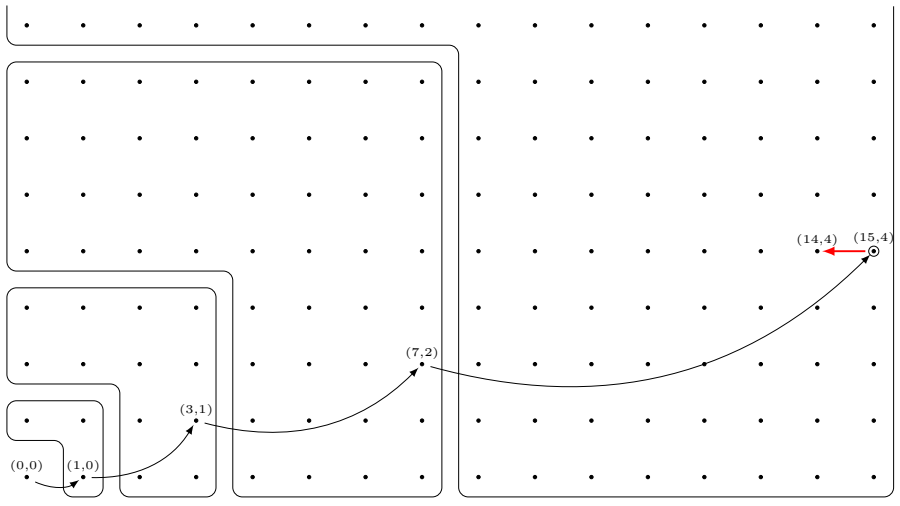
Algorithme 2 PARCOURS

Préconditions: $w \in \mathcal{F}^*$ où $|w| = n$, $\boxed{\text{o}} \in \mathcal{N}$ initialisé jusqu'à la racine.

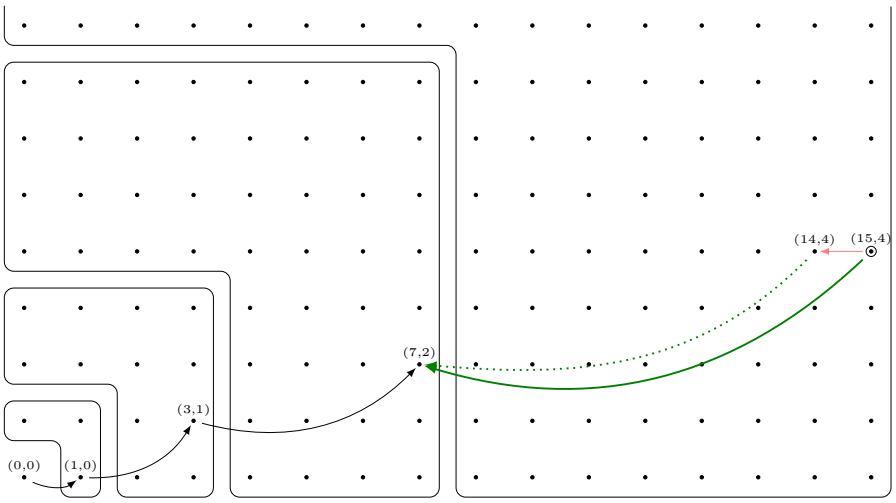
Postconditions: $\boxed{r} \in \mathcal{N}$ le dernier noeud visité après la lecture de w.

1: **function** PARCOURS$\left(w, \boxed{o}\right)$
2: incrémenter Σ de $\boxed{o}$.
3: **si** |w| = 0 **alors**
4: $\boxed{r} \leftarrow \boxed{o}$
5: **sinon**
6: $\boxed{d} \leftarrow$ VOISIN $\left(\boxed{o}, \Delta\right)$
7: étiqueter explicite le lien de voisinage V_Δ de $\boxed{o}$
8: étiqueter explicite le lien de voisinage $V_{\bar{\Delta}}$ de $\boxed{d}$
9: $\boxed{r} \leftarrow$ PARCOURS $\left(w_{1..n}, \boxed{d}\right)$
10: **retourner** $\boxed{r}$
11: **fin function**

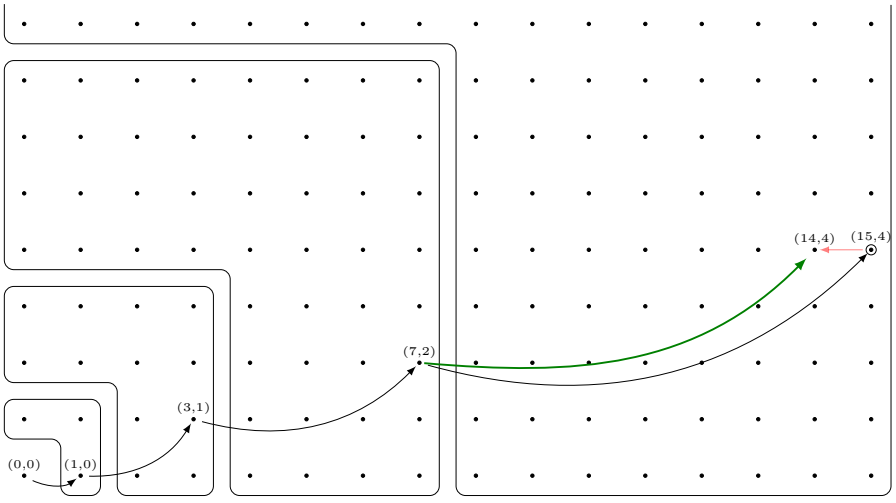
1.



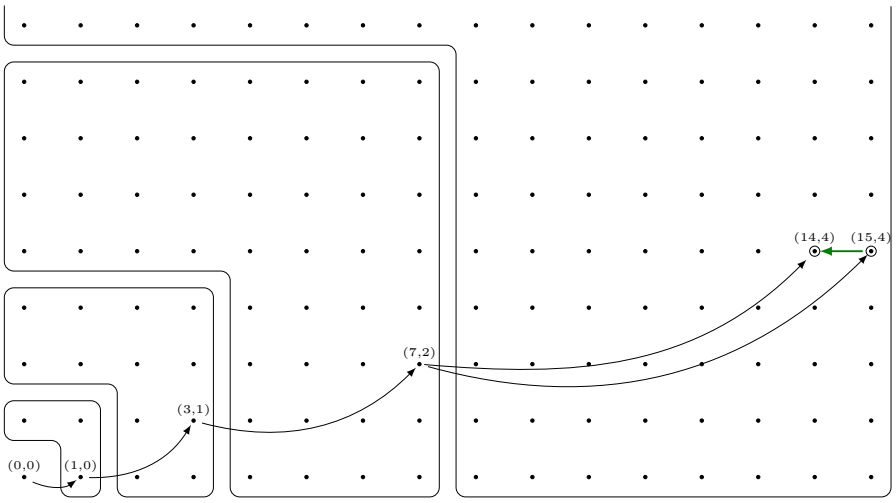
2.



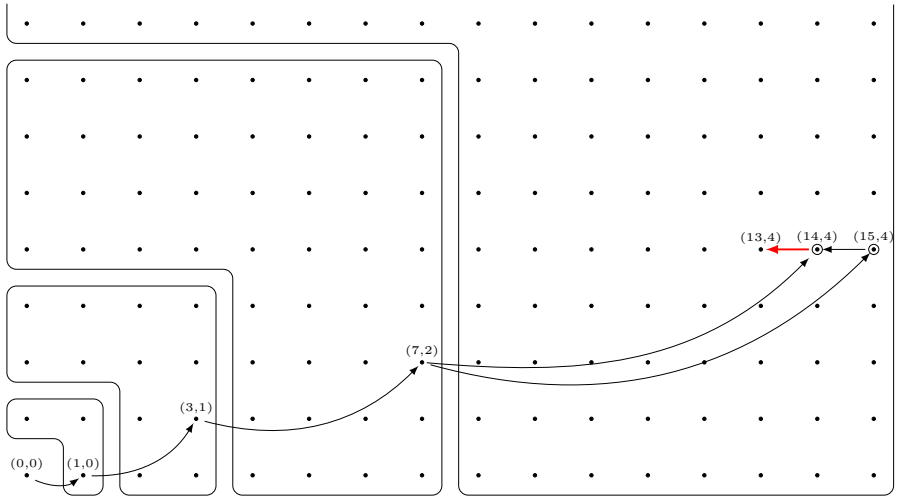
3.



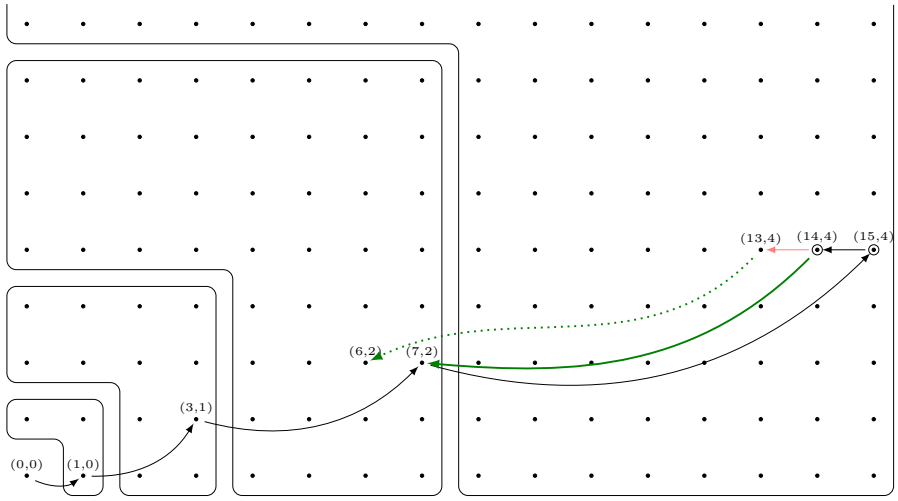
4.



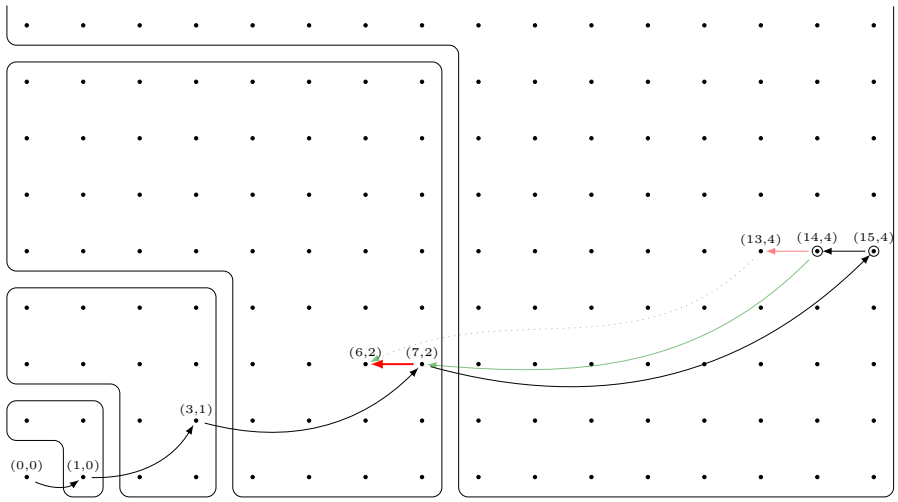
5.


(13,4)
(14,4)
(15,4)
(7,2)
(3,1)
(0,0)
(1,0)

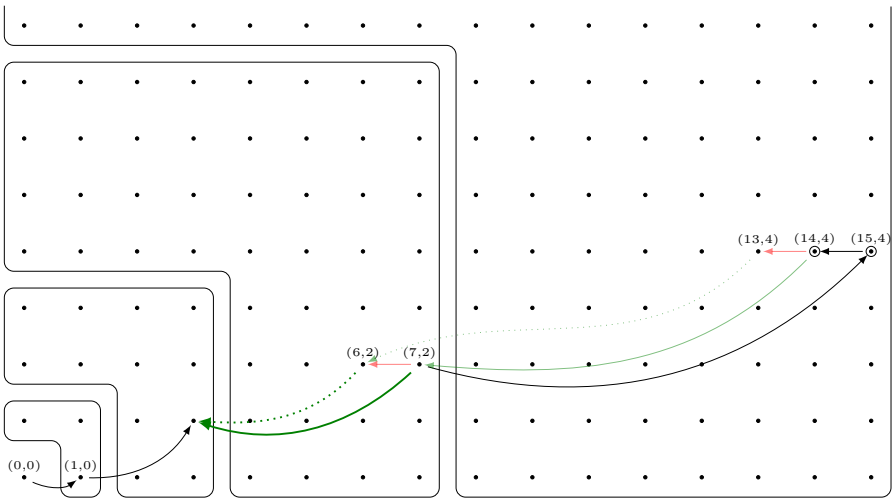
6.


(13,4)
(14,4)
(15,4)
(6,2)
(7,2)
(3,1)
(0,0)
(1,0)

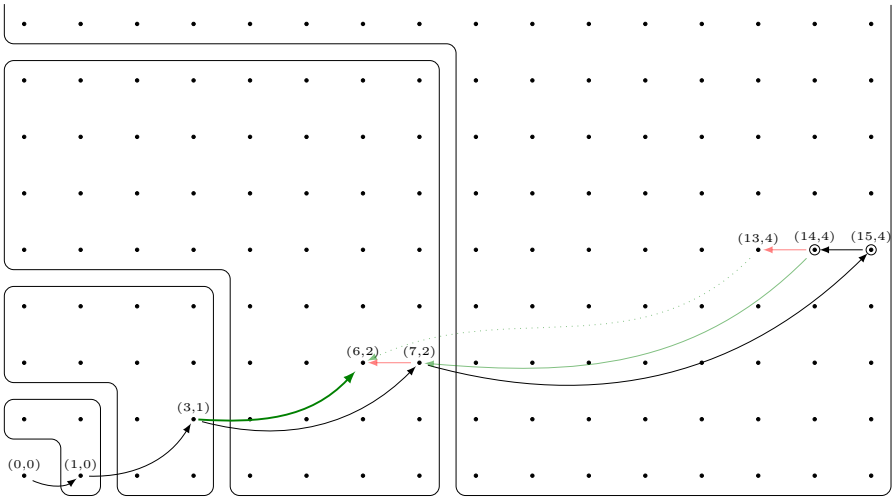
7.


(13,4)
(14,4)
(15,4)
(6,2)
(7,2)
(3,1)
(0,0)
(1,0)

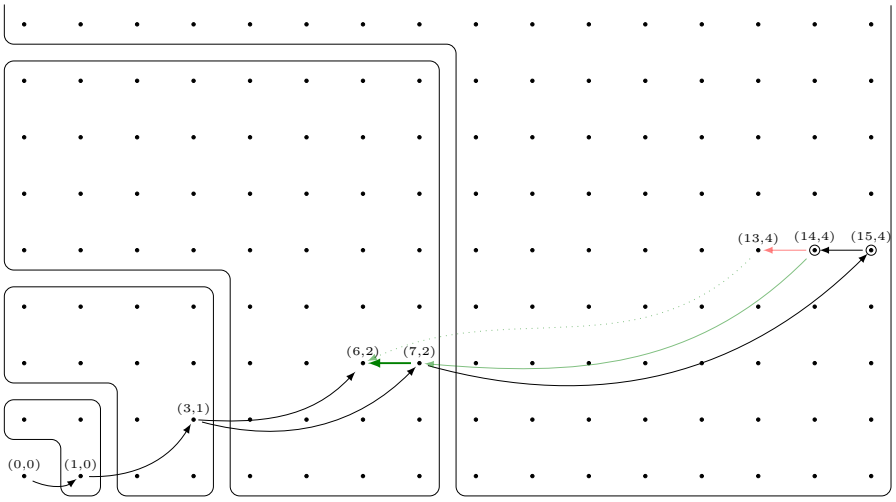
8.


(13,4)
(14,4)
(15,4)
(6,2)
(7,2)
(0,0)
(1,0)

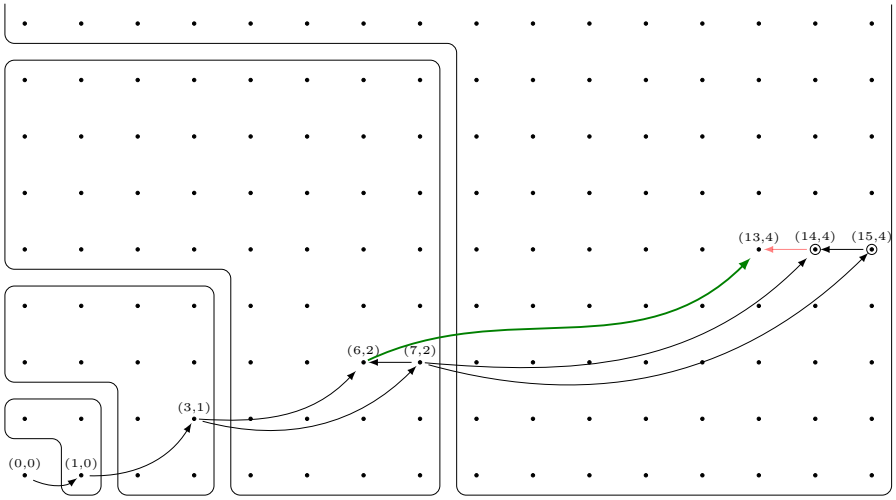
9.


(13,4)
(14,4)
(15,4)
(6,2)
(7,2)
(3,1)
(0,0)
(1,0)

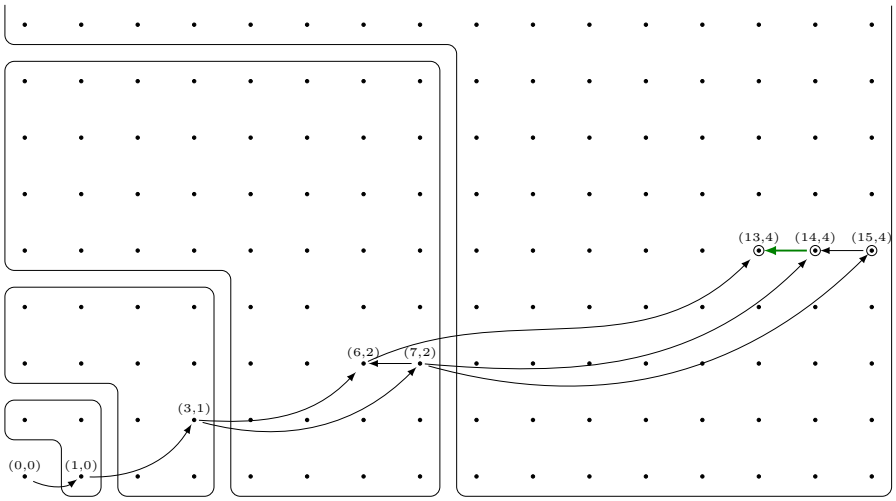
10.


(13,4)
(14,4)
(15,4)
(6,2)
(7,2)
(3,1)
(0,0)
(1,0)

11.



12.



Voyons maintenant comment étendre cette structure aux autres quadrants du plan. L'astuce consiste à remplacer la racine de l'arbre par un noeud virtuel, α qui n'appartient pas à $\mathbb{Z}^2$ mais a quatre enfants, un dans chaque quadrant,

$$e\left(\boxed{\alpha}\right) = \left(\boxed{(0,0)}, \boxed{(0,-1)}, \boxed{(-1,-1)}, \boxed{(-1,0)}\right) \tag{2.4}$$

Pour détecter un enfant de α, on utilise la propriété $f(d) = d$, vraie si et seulement si d est un enfant de α. Il suffit ensuite de faire une légère modification à l'algorithme VOISIN tel qu'indiqué en rouge dans l'algorithme 3. On note que les coordonnées, les voisins et le père de α sont indéfinis, mais que l'algorithme n'y accède jamais.

Avant d'aborder le théorème de linéarité de PARCOURS, on remarquera d'abord qu'on

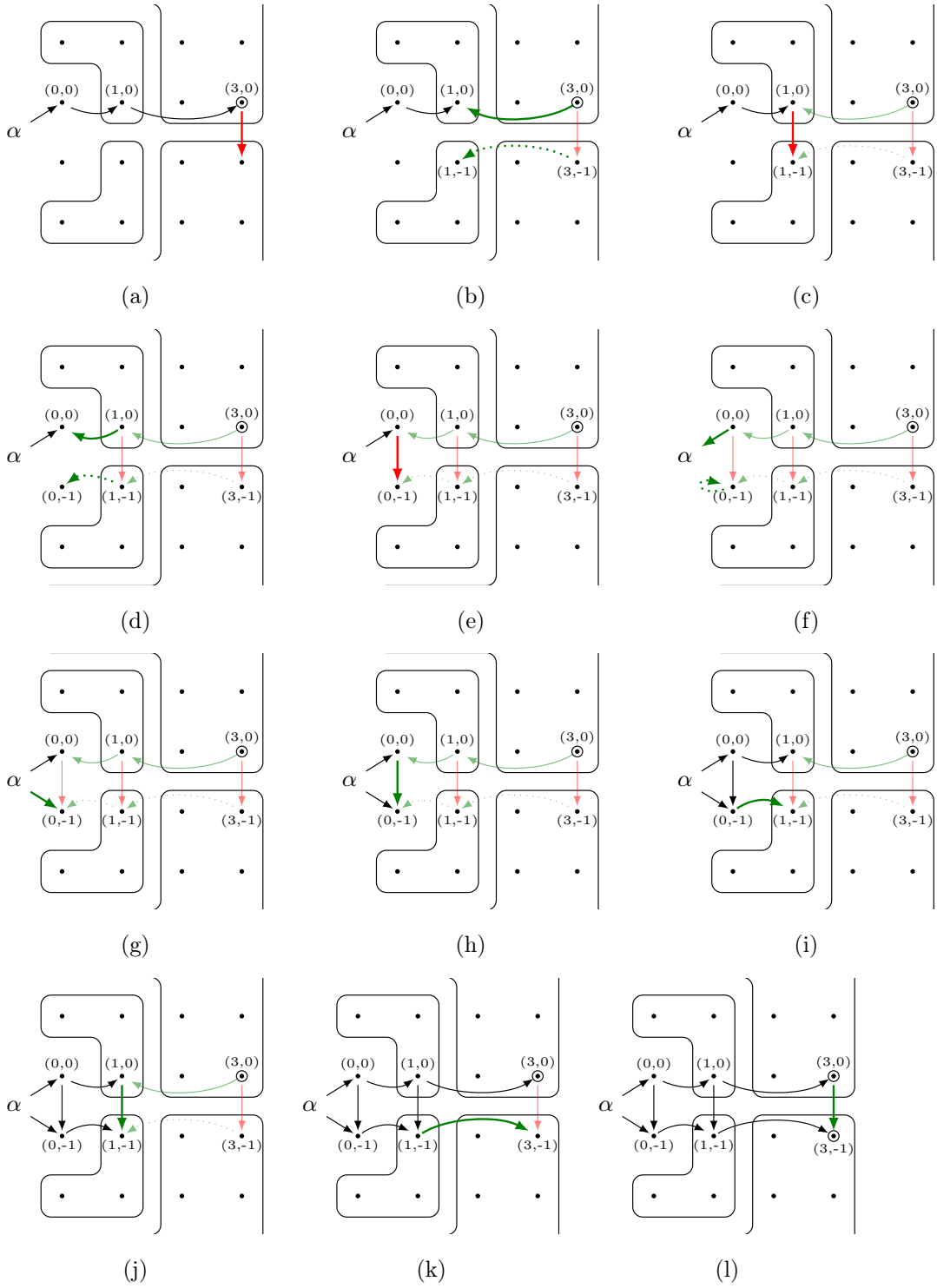


Figure 2.10: Changement de quadrant

Algorithme 3 VOISIN étendu au plan

Préconditions: $\boxed{o} \in \mathcal{N}$, $\Delta \in \mathcal{F}$

Postconditions: $\boxed{o} \xrightarrow{\Delta} \boxed{\text{d}}$ correctement initialisé

1: **function** VOISIN$\left(\boxed{o}, \Delta\right)$
2: **si** $\boxed{o} \xrightarrow{\Delta} \boxed{\text{d}}$ n'est pas initialisé **alors**
3: $d \leftarrow o + \Delta$
4: **si** d est un fils de o **alors**
5: $\boxed{d} \leftarrow$ ENFANT $\left(\boxed{o}, d\right)$
6: **sinon**
7: $\boxed{\text{pere}} \leftarrow$ PERE $\left(\boxed{o}\right)$
8: **si** $f(d) = d$ ou $f(d) = pere$ **alors**
9: $\boxed{d} \leftarrow$ ENFANT $\left(\boxed{\text{pere}}, d\right)$
10: **sinon**
11: $\boxed{d} \leftarrow$ ENFANT $\left(\text{VOISIN}\left(\boxed{\text{pere}}, \Delta\right), d\right)$
12: initialiser $\boxed{o} \xrightarrow{\Delta} \boxed{d}$
13: initialiser $\boxed{d} \xrightarrow{\bar{\Delta}} \boxed{o}$
14: **retourner** $\boxed{d}$
15: **fin function**

peut compter les ancêtres de $\boxed{\text{n}} \in \mathcal{N}$ grâce à la fonction f

$$\left|\bigcup_{i=1}^{h} \text{PERE}^i\left(\boxed{\text{n}}\right)\right| = \left|\bigcup_{i=1}^{h} f^i(n)\right| + 1 \tag{2.5}$$

ainsi on pourra effectuer les démonstrations dans $\mathbb{Z}^2$ plutôt que dans $\mathcal{N}$. De plus, il sera utile d'étendre f aux ensembles de points et finalement, nous aurons besoins du lemme suivant pour borner le nombre de noeuds créés.

Lemme 2. *Soit* $E = \{p_0, p_1, p_2, p_3, p_4\}$ *où* $p_i \in \mathbb{Z}^2$ *et* $w \in \mathcal{F}^4$ *tel que pour* $i = 0, 1, 2, 3$ *on a* $p_{i+1} = p_i + w_i$, *alors* $|f(E)| \leq 4$.

Démonstration. L'arbre radix divise le plan en sections de 2×2 points partageant le même père. Par conséquent, au moins deux p_i partagent le même père, d'où la borne $|f(E)| \leq 4$. □

Théorème 1. (Brlek *et al.*, 2011). *L'algorithme* PARCOURS *est linéaire.*

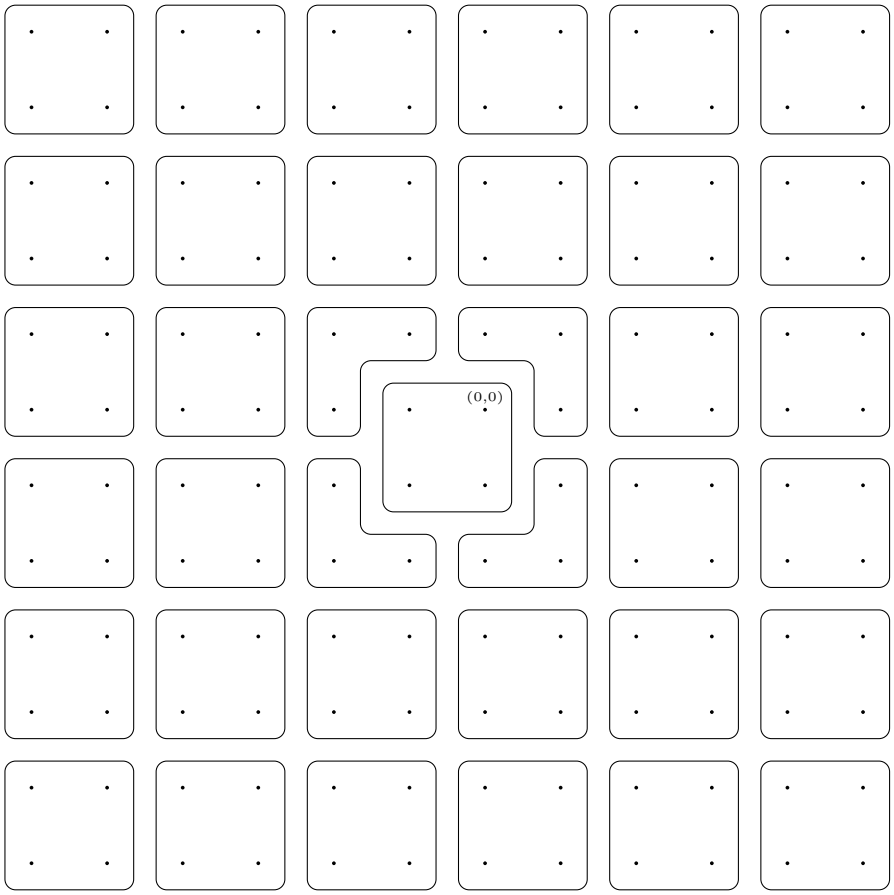


Figure 2.11: Les points du plan regroupés par parent

Démonstration. On remarque d'abord que l'essentiel du temps passé dans VOISIN est causé par l'appel récursif, puisque toutes les autres opérations sont d'ordre constant. Donc VOISIN est d'ordre $\mathcal{O}(\mathrm{h})$ où h est la hauteur de l'arbre. Cependant, un appel récursif n'est effectué sur un noeud que la première fois où l'un de ses enfants recherche son voisin, le résultat étant stocké directement au niveau de l'enfant la suite. Ainsi, pour tout noeud, VOISIN est appelé un maximum de 16 fois (les quatre voisins de ses quatre enfants). Le temps total passé dans VOISIN est donc proportionnel au nombre de noeuds créés. Reste à montrer que PARCOURS créé un nombre de noeuds proportionnel à $|w|$.

Soit N tous les noeuds créés et $N_v \subseteq N$ les noeuds visités. On va montrer que le nombre total de noeuds créés est borné par un multiple du nombre de noeuds visités. Clairement, $|N_v| \leq |N|$ et $|N_v| \leq |w|$. De plus, on obtient N à partir de N_v grâce à f, $N = \bigcup_{i=0}^{h} f^i(N_v)$ donc

$$|N| = \sum_{i=0}^{h} \left| f^i(N_v) \right|. \tag{2.6}$$

Par construction chaque noeud ajouté à N_v est voisin de son précédent. Le lemme 2 s'applique et on regroupe les éléments par 5

$$|f(N_v)| \leq 4 \left\lceil \frac{|N_v|}{5} \right\rceil \leq \frac{4}{5}(|N_v| + 4) \tag{2.7}$$

Le lemme 1 nous permet d'appliquer récursivement le lemme 2 sur f. On combine donc les équations 2.6 et 2.7 pour obtenir l'expression

$$|N| \leq \sum_{i=0}^{h} \left| f^i(N_v) \right| \leq \sum_{i=0}^{h} \left(\left(\frac{4}{5} \right)^i |N_v| + \sum_{j=0}^{i} \left(\frac{4}{5} \right)^j 4 \right) \leq \cdots \tag{2.8}$$

qu'on borne ensuite à l'aide de l'identité $\sum_{i=1}^{\infty} \left(\frac{4}{5}\right)^i = 5$ pour finalement avoir

$$\cdots \leq 5|N_v| + 4 \sum_{i=0}^{h} 5 \leq 5|N_v| + 20h \tag{2.9}$$

Puisque h est précisément le nombre de bits requis pour écrire les coordonnées des points dans N, $h \in \mathcal{O}(\log n)$ et ainsi $|N| \in \mathcal{O}(n)$. Finalement, puisque $N_v \leq |w|$ on a que PARCOURS $\in \mathcal{O}(|w|)$, tel que voulu. □

Méthode	Complexité	Commentaires
Adressage direct	$\mathcal{O}(n^2)$	• Rapide pour les petites valeurs de n. • Structure de données native en informatique. • On doit connaître n au départ.
Tri	$\mathcal{O}(n \log n)$	• Méthode la moins gourmande en mémoire. • On n'a pas à connaître n au départ. • Structures de données déjà implémentées sous forme de librairies.
BKP	$\mathcal{O}(n)$	• On n'a pas à connaître n au départ. • Maintient sa performance même avec les chemins très longs. • Plus lent que les autres méthodes pour les chemins courts.

Tableau 2.2: Récapitulation des méthodes de détection d'intersection

Le tableau 2.2 récapitule les avantages et les désavantages des trois méthodes de détection d'intersection présentées pour un chemin w de longueur $|w| = n$

On en arrive à l'algorithme principal de ce chapitre.

On commence à la ligne 2 par construire l'arbre radix augmenté associé au mot w. En commençant au point $c = W$, on procède pour chaque position c

1. Extraire la lettre $\Delta \in \mathcal{F}$ associée au vecteur $\vec{cv}$ pour chaque voisin v de c.
2. Déterminer le virage associé à chaque Δ.
3. Choisir le virage le plus à droite, c'est à dire celui le plus proche de 3.

L'algorithme se termine lorsqu'on revient au point W.

Algorithme 4 Algorithme HULL

Préconditions: Un mot $w \in \mathcal{F}^*$ codant un chemin discret.

Postconditions: Un mot $w' \in F^*$ codant Hull(w).

function HULL(w)
    Invoquer PARCOURS sur le mot w à partir de $(0, 0)$.
    Soit W un point de la factorisation standard.
    Soit V l'ensemble des voisins explicites de W.
    $c \leftarrow V_0$
    $w' = \text{Step}(c - W)$
    tant que $c \neq W$ ou $|V| \neq 0$ **faire**
        $turn \leftarrow 2$
        pour chaque voisin explicite v de c **faire**
            $\Delta \leftarrow \text{Step}(v - c)$
            si $\left(\Delta - w'_{n-1}\right) + 1 \mod 4 \leq turn + 1 \mod 4$ **alors**
                $turn \leftarrow \Delta - w'_{n-1}$
                $suivant \leftarrow v$
        $w' = w' \cdot \Delta$
        Retirer c de V
        $c \leftarrow suivant$
    retourner w'
fin function

Théorème 2 (Justesse de l'algorithme 4.)**.** *Pour chaque mot $w \in \mathcal{F}^*$ l'algorithme 4 calcule Hull(w).*

Démonstration. Soit Hull(w) de longeur $k \in \mathbb{N}$. On utilise l'invariant de boucle suivant :

« Au début de la i-ième itération de la boucle à la ligne 7, w' est un préfixe de longeur i du mot de contour associé à Hull(w). »

L'invariant tient lors de la première itération de la boucle à la ligne 7, puisqu'à ce moment w' a été initialisé à la ligne 6. On suppose donc que l'invariant tient encore au début de la i-ième itération. w' est alors un préfixe de longeur i de Hull(w). Les lignes 9 à 13 calculent la direction du voisin le plus à droite et la ligne 14 concatène cette direction à w'. Par la règle de la main droite, choisir le voisin le plus à droite assure de demeurer sur l'enveloppe extérieure de w. Ainsi, à la fin de la i-ième itération, w' est un préfixe du mot de contour associé à Hull(w) de longueur $i+1$. Finalement, à la fin de la boucle, w' est un préfixe de Hull(w) de longueur k, autrement dit $w' = \text{Hull}(w)$. Notons que puisque tous les voisins de W sont sur l'enveloppe externe, la ligne 15 retire bien tous les éléments de V ce qui garantit que l'algorithme se termine. □

Nous terminons maintenant cette section en démontrant que l'algorithme 4 est linéaire en temps et en espace. On a déjà démontré que PARCOURS est linéaire en temps. On peut facilement obtenir W lors du parcours initial sans aucun coût supplémentaire. Les lignes 3, 4, 5 et 6 s'exécutent toutes en temps constant. Ensuite, la boucle extérieure à la ligne 7 est exécutée exactement k fois, une fois pour chaque lettre de w' et $|w'| \leq |w|$. Pour chaque itération de la boucle extérieure, la boucle intérieure s'exécute un maximum de 4 fois. Le temps total de la boucle est donc de $k(4c_1 + c_2)$, où c_1 et c_2 sont des constantes appartenant à $\mathbb{R}$, ce qui montre bien que l'algorithme 4 dans son ensemble est linéaire en temps. On termine la preuve en remarquant que l'arbre radix augmenté est linéaire en espace et qu'aucun nouveau noeud n'est créé après l'appel initial à PARCOURS.

CHAPITRE III

GÉNÉRATION EXAUSTIVE

Dans ce chapitre nous présentons un algorithme de génération exhaustive basé sur le parcours du bord extérieur d'un polyomino.

Si nous connaissons le polyomino à l'avance, il existe un parcours canonique de son bord de longueur minimale. Pour l'obtenir nous avons besoin de deux choses : un point de départ canonique sur le bord et une orientation du parcours. Ensuite, à partir de ce point de départ, on se déplace de manière à longer le bord du polyomino.

Nous présentons la classe des polyominos ainsi énumérés ainsi que leur représentation à l'aide de pierres de *go* dans la section 1. Dans la section 2, nous décrirons un jeu basé sur le *go* dont une partie terminée représente un polyomino délimité par un nombre donné de pierres noires. Nous terminerons le chapitre avec la section 3, où nous présentons l'algorithme de génération exhaustive ainsi qu'une spécialisation pour les polyominos avec différents types de trou.

3.1 Les gominos

On considère $\mathbb{Z}^2$ comme une planche de *go* infinie et on y représente le contour extérieur P^+ d'un polyomino P à l'aide de pierres noires, et le contour intérieur P^- à l'aide de pierres blanches tel qu'illustré à la figure 3.1.

Définition 13. *Un* gomino *est une paire $G = (P^+, P^-)$ d'ensembles dans $\mathbb{Z}^2$ pour lesquels il existe un polyomino P tel que P^+ est son contour extérieur et P^- est son contour intérieur.*

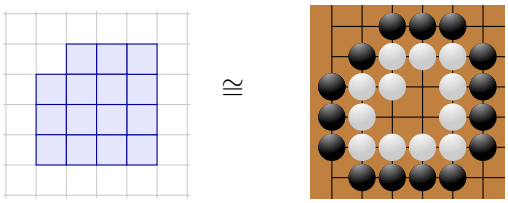


Figure 3.1: Un polyomino simplement 4-connexe et son gomino.

L'analogie avec le *go*, autre que le fait d'utiliser des pierres noires et blanches, réside dans le fait que selon la terminologie du *go*, un polyomino simplement 4-connexe P avec son contour P^+ est le *territoire noir*, et P^+ délimite ce territoire de façon minimale. L'algorithme décrit plus loin génère chaque gomino avec n pierres noires, donc chaque polyomino de périmètre de site n, en plaçant tour à tour des pierres noires et blanches, comme le feraient deux joueurs de *go*.

3.2 Un jeu bien solitaire

Le jeu de *Gomino* est d'une certaine façon une version simplifiée du jeu de *go*. D'une position initiale, deux joueurs **B** et **W** jouent sur une planche de jeu $\mathbb{N} \times \mathbb{Z}$ en laissant des pierres sur chaque point de la grille visité et libre jusque là. **B** a n pierres noires en main et tente d'encercler un territoire en revenant à sa position d'origine, tandis que **W** essaie de l'en empêcher en plaçant des pierres blanches.

On décrit la séquence des positions successives de chacun des joueurs à l'aide deux fonctions

$$\mathbf{B}, \mathbf{W} : \mathbb{N} \to \mathbb{N} \times \mathbb{Z},$$

dont on fixe les points de départ

$$\mathbf{B}(0) = (0, 0) \quad ; \quad \mathbf{W}(0) = (1, 0). \tag{3.1}$$

La notion suivante est utile pour décrire les 4-trous.

Définition 14. *Un* motif en X *est un alignement de pierres* 2×2 *tel que l'une de ses diagonales est noire, et l'autre blanche.*

Figure 3.2: Motifs en X

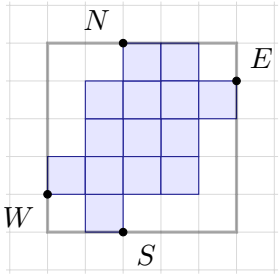
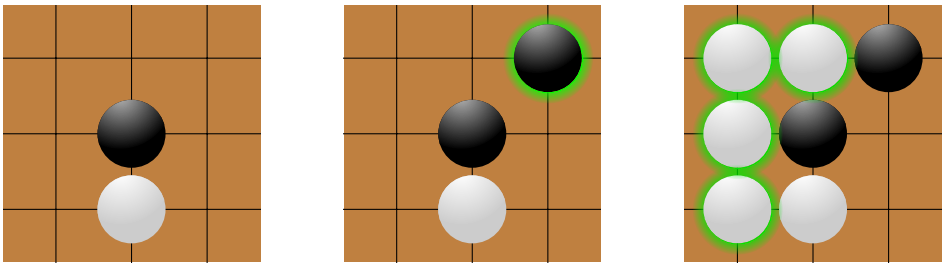


Figure 3.3: Factorisation NSEW du contour

3.2.1 Position de départ et zones interdites

Afin de rendre chaque parcours unique et éviter la répétition de polyominos par translation, on choisit une cellule du polyomino P, appellée la *racine* de P. L'une des manières d'obtenir canoniquement une telle cellule est d'utiliser la factorisation standard tel qu'illustré à la figure 3.3 et de choisir la cellule ayant comme sommet le point W.

Pour s'assurer de respecter les contraintes imposées par le choix de la racine, on interdit à **B** de jouer sous sa position initiale en colonne 0 et à **W** de jouer sur la colonne 0. Ces positions sont appelées *zones interdites*.



(a) Position initiale (b) Coup de **B** (c) Réplique de **W**

Figure 3.5: L'anatomie d'un coup

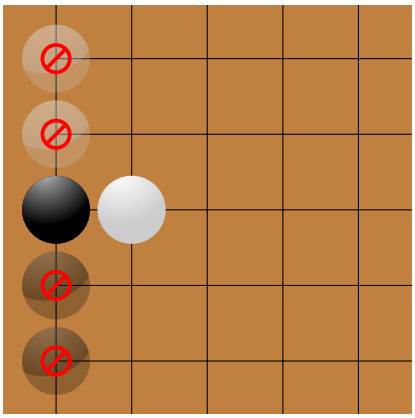


Figure 3.4: Position de départ et zones interdites

3.2.2 Règles de déplacement

Le joueur **B** commence, et les joueurs jouent à tour de de rôle (figure 3.5).

La liste des coups possibles est illustrée à la figure 3.6. La pierre noire marquée en vert indique la position de **B**, tandis que la pierre blanche marquée en vert qui y est 4-connexe indique celle de **W**. Les règles sont les suivantes :

B : ce joueur se déplace exactement d'un pas dans son 8-voisinage excluant la zone interdite et les positions déjà occupées par des pierres blanches. On remarque que seuls les déplacements vers des intersections libres augmentent le nombre de

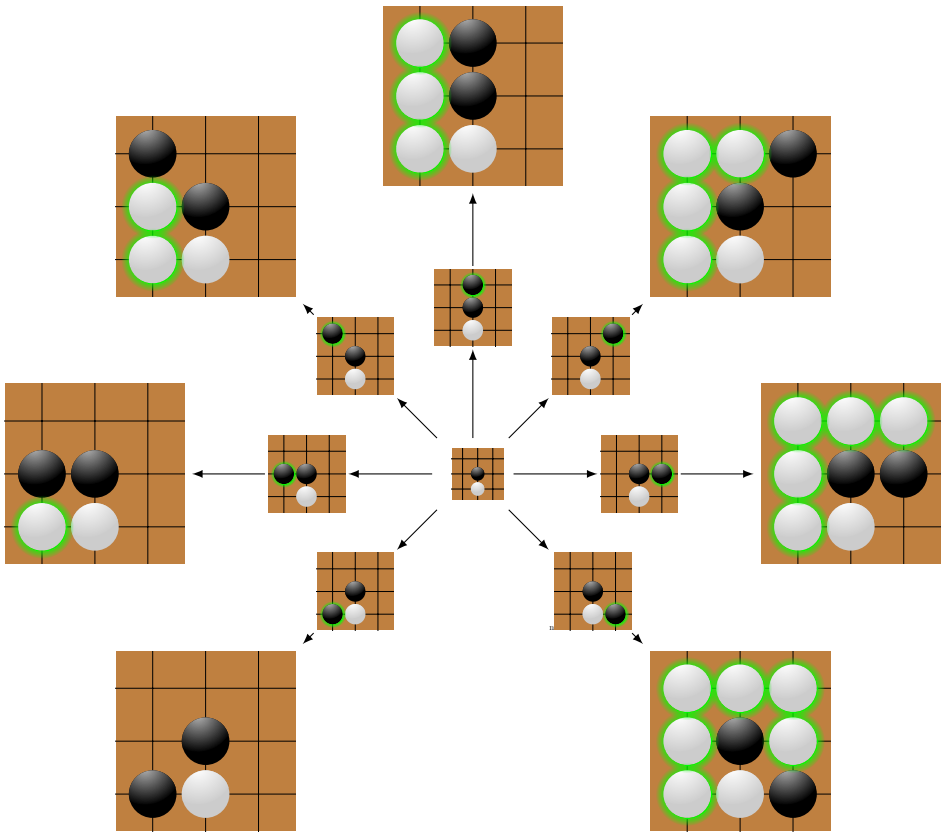


Figure 3.6: Coups permis (à rotation près)

pierres noires en jeu. De plus, **B** doit jouer son premier coup en direction **7**. Cette contrainte force **B** à parcourir le bord du polyomino en sens anti-horaire.

W : ce joueur doit terminer son coup dans le 4-voisinage de **B** en un nombre minimal de pas, de 0 à 6 selon le coup de **B**. Il peut le faire en effectuant autant de pas qu'il le faut sur les intersections vides ou sur les pierres blanches dans le 8-voisinage de la position précédente de **B** tout en respectant les conditions suivantes :

1. *Règle de la main droite.* À la fin de chaque tour, **W** doit être situé à gauche de la flèche décrivant le dernier coup de **B**.

   Cette règle force **W** à rester à l'intérieur du polyomino dont le bord extérieur est construit par **B**.

2. *Règle des trous.* **W** ne peut en aucun cas former de motif en X.

   La règle des trous évite de créer des polyominos contenant des 8-trous et ainsi assure que tous les polyominos générés par l'algorithme seront simplement 4-connexe. Cette contrainte sera relâchée plus tard lorsque nous adapterons l'algorithme pour obtenir les polyominos contenant les différents types de trous.

Pour comprendre l'ajout de plusieurs pierres blanches en un seul coup, on doit se rappeler que par définition, **B** encercle **W** avec un minimum de pierres. Chaque coup de **B** nous révèle donc un peu plus d'information sur **W**. En particulier, si **W** peut jouer une pierre alors on sait que cet emplacement fait bien partie de **W** puisque dans le cas contraire **B** aurait tout simplement joué là au tour précédent et ainsi bloqué **W**.

On remarque aussi que la séquence qui mène à l'ajout de 6 pierres est interdite par la règle des trous et donc n'est pas un coup valide pour **W**. Elle sera par contre utilisée plus loin pour générer les polyominos simplement 8-connexes.

Finalement, la restriction de **W** sur la colonne 0 garanti que si **B** joue sur la colonne -1 on a une victoire de **W**. Puisqu'on cherche à énumérer les victoires de **B**, on évitera d'y jouer.

3.2.3 Fin de partie

On dit que **B** *gagne* s'il parvient à revenir à sa position d'origine. Dans ce cas, les chemins parcourus par **B** et **W** décrivent respectivement le contour extérieur et intérieur d'un unique polyomino P.

Si à n'importe quelle étape, **W** est incapable de rejoindre le 4-voisinage de **B** en utilisant un coup permis, alors **W** gagne. De plus, quand aucune séquence de coups ne mène à une victoire de **B**, alors **W** gagne. En conséquence, il y a bijection entre les polyominos et les parties de Gomino gagnées par **B**.

Théorème 3. *Tout polyomino simplement* 4*-connexe* P *est décrit par une unique partie de gomino* G *gagnée par* **B**.

Démonstration. Soit P un polyomino simplement 4-connexe, P^+ et P^- ses contours extérieurs et intérieurs. On construit, par induction, une partie de Gomino qui décrit P et on montre que cette partie est unique.

On débute avec $\mathbf{B}(0)$ et $\mathbf{W}(0)$ tel que décrit par l'équation (3.1), et on identifie $\mathbf{W}(0)$ à la racine de P. Puisque le premier coup de $\mathbf{B}$ doit absolument être 0, on a que $\mathbf{B}(1) = (1, -1)$, qui est un point de P^+, et $\mathbf{W}(1) = (1, 0)$, qui est un point de P^-.

Supposons maintenant que $\mathbf{B}$ a déjà joué n coups tels que chaque pierre noire placée jusqu'à maintenant est dans P^+ et chaque pierre blanche est dans P^-. Soit $\vec{b}_n$ la flèche de $\mathbf{B}(n-1)$ vers $\mathbf{B}(n)$. À l'étape n, nous sommes, à rotation près, dans la situation illustrée à la figure 3.7. Certaines intersections peuvent contenir des pierres. On numérote les intersections en commençant à 0 à partir de $\mathbf{W}(n)$.

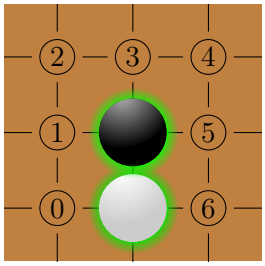


Figure 3.7: Ordre du 8-voisinage

On remarque qu'en numérotant le voisinage de $\mathbf{B}(n)$ en sens horaire en commençant à $\mathbf{W}(n)$, le point $\mathbf{B}(n+1)$ est le premier point de P^+ rencontré. En effet, lorsque $\mathbf{B}$ place une pierre en i, $\mathbf{W}$ réplique en plaçant ses pierres en $1, 2, \ldots, i-1$, ce qu'il peut toujours faire puisque ces positions sont libres et dans le 8-voisinage de $\mathbf{B}(n)$. S'il existait une position $j < i$ contenant déjà une pierre noire, alors $\mathbf{W}$ serait incapable de se connecter à la nouvelle position de $\mathbf{B}$ et serait victorieux donc la partie ne représenterait pas P. De même, s'il existe une position $j < i$ qui contiendra éventuellement une pierre noire, $\mathbf{B}$ doit obligatoirement jouer sur ce point de P^+, sinon $\mathbf{W}$ y placera une pierre blanche et notre gomino ne représentera plus P. □

Remarque : Une conséquence directe du théorème 1 est que la séquence des coups joués par **B** lors d'une victoire, codée par un mot sur $\mathcal{A}$, est suffisante pour décrire un polyomino unique. Par exemple, le polyomino illustré à la figure 3.1 est encodé par le mot $w =$ 700012223445566.

3.3 L'algorithme de génération

Nous utilisons maintenant la bijection entre les parties de Gomino et les polyominos pour décrire notre principal algorithme qui permet la génération exhaustive des polyominos simplement 4-connexes. Nous parlerons simplement de *l'algorithme des gominos*. Lorsque nous faisons référence à une partie de gomino, celle-ci peut être inachevée.

L'algorithme des gominos. *L'algorithme accepte en entrée une partie de gomino de n coups et retourne en sortie la liste des parties de $(n+1)$ coups possibles à partir de ce point.*

1. Construire la liste des coups valides

   D'abord le coup de **B** *: toute intersection vide ou contenant une pierre noire dans le 8-voisinage de la position actuelle de* **B** *est un coup valide.* **W** *répond immédiatement et comme il est montré à la figure 3.6, a un seul coup possible. Si* **W** *est incapable de jouer, rejeter ce coup immédiatement. Sinon, l'ajouter à la liste.*

2. Créer de nouvelles parties

   Pour chaque coup valide, créer une copie de la partie actuelle, y jouer ce coup et ajouter la partie à l'ensemble des nouvelles parties. Pour chaque partie de n coups, l'algorithme peut créer jusqu'à sept nouvelles parties de $(n+1)$ coups. La partie reçue en paramètre par l'algorithme n'est plus utilisée par la suite et peut être effacée à ce moment.

3. Comptabiliser les victoires

   Les conditions de victoires sont vérifiées sur chaque nouvelle partie. Les victoires de **W** *sont rejetées et les victoires de* **B** *sont conservées (ce sont nos polyominos). Puis les autres parties sont ajoutées à l'ensemble des parties en cours.*

4. Appliquer récursivement

   On applique ensuite l'algorithme récursivement sur chacune des parties en cours, jusqu'à ce que cet ensemble soit vide. Puisque chaque partie est indépendante des autres, cette étape peut être traitée en parallèle.

3.3.1 4-trous

Nous présentons maintenant une extension de l'algorithme de base qui permet de générer les polyominos simplement 8-connexes en retirant la règle des trous. Le coup blanc `422006` devient alors valide et décrit un trou d'aire 1.

Notons que la preuve du théorème 3 ne fait aucune mention de 4-trous. Conséquemment, le théorème reste vrai pour les polyominos simplement 8-connexes incluant des motifs en X. Bien que ces motifs soient permis pour les polyominos simplement 8-connexes, nous pouvons les utiliser pour compter les 4-trous.

Proposition 2. *Si un gomino G décrit un polyomino simplement* 8*-connexe P, alors il y a autant de motif en X dans G qu'il y a de* 4*-trous dans P.*

Démonstration. Procédons par induction sur le nombre de 4-trous dans un polyomino P. Considérons tout d'abord le cas d'un seul 4-trou. Il y a au moins un motif en X, puisque c'est le seul motif qui bloque tout 4-chemin de l'intérieur vers l'extérieur du trou tout en permettant l'existence de 8-chemin. S'il y avait un 4-chemin de l'intérieur vers l'extérieur, alors ce ne serait pas un trou. D'un autre coté, si on suppose l'existence de deux motifs en X, alors il existe deux 8-chemins reliant l'intérieur du trou à l'extérieur. Prenons deux points, par exemple z à l'intérieur du trou, et w à l'extérieur. On peut relier z et w par deux chemins différents qui ne croisent pas P, chacun passant par l'un des motifs en X. Ces deux chemins forment un 8-chemin fermé contenant une partie de P (la *partie intérieure*). Une autre partie de P est à l'extérieur du chemin fermé (la *partie extérieure*), sinon z ne serait pas à l'intérieur d'un 4-trou. Finalement, il n'existe pas de 4-chemin qui relie la partie intérieure à la partie extérieure sans croiser le 8-chemin donc P n'est pas 4-connexe : une contradiction. Il n'y a donc qu'un seul motif en X.

Supposons maintenant que pour tout polyomino simplement 8-connexe avec $(n-1)$ 4-trous, il y a exactement $(n-1)$ motif en X. Soit P un polyomino simplement 8-connexe avec n 4-trous. Il y a au moins un 4-trou qui peut être rempli afin d'obtenir le polyomino simplement 8-connexe P'. Nous appellerons ce 4-trou le $n^{\text{ième}}$ trou. Le polyomino P' a exactement $(n-1)$ 4-trou, alors par hypothèse d'induction il possède aussi $(n-1)$ motifs en X. Par le même raisonnement que pour le cas initial, l'ajout du $n^{\text{ième}}$ trou ajoute exactement un motif en X. P a donc n motifs en X, tel que voulu. □

3.3.2 8-trous

Pour obtenir les 8-trous, nous allons utiliser une version légèrement modifiée de l'algorithme de base sur les positions libres à l'intérieur d'un gomino. Ces positions devront être remplies par des pierres noires ou blanches jusqu'à ce que **W** n'ait plus de *libertés* pour réutiliser la terminologie du *go*, c'est à dire qu'il ne reste plus aucune intersection libre dans le 4-voisinage des pierres blanches. On remarquera qu'il n'y a jamais d'intersections libres dans les 4-trous 8-connectés à l'extérieur : ces trous sont déjà générés correctement par l'algorithme étendu présenté à la section 3.3.1.

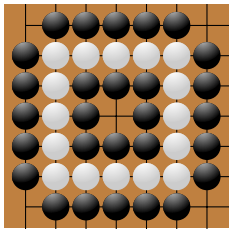
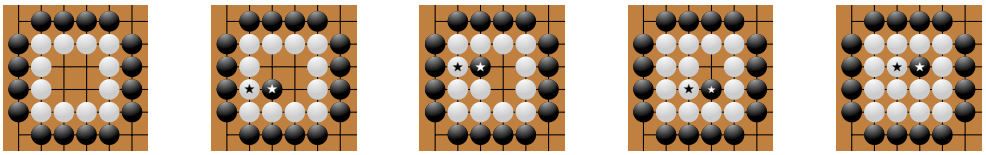


Figure 3.8: Un gomino sans libertés pour **W**

Comme pour l'algorithme des gominos, les trous sont énumérés en fixant leur cellule en bas à gauche. Pour ce faire, on exécute l'algorithme avec chaque position de départ possible, de bas en haut et de gauche à droite. Une position de départ valide pour le **B** est à droite d'une pierre blanche. Cette pierre blanche est la position initiale de **W**. **B** gagne si les deux joueurs reviennent à leurs positions initiales. Ceci garantit que le chemin suivi par **W** est fermé, et ainsi délimite bien un 8-trou. Cette fois cependant, **W**

peut dépasser son point de départ en réponse à **B**. Ainsi, **B** gagne dès que **W** revient à son point de départ, même si **W** n'as pas terminé son coup.



(a) A gomino (b) Position 1 (c) Position 2 (d) Position 3 (e) Position 4

Figure 3.9: Les quatres positions initiales pour énumérer les trous de 3.9a

En plus des nouvelles positions de départ et conditions de fin, nous devons ajouter un nouveau coup pour **B** afin de pouvoir générer tous les trous. Ainsi, à son premier coup, **B** peut maintenant gagner immédiatement en restant sur place. **W** est alors forcé de faire un tour complet autour de **B**, terminant la partie. Ce coup est nécessaire et suffisant pour produire les trous d'une seule pierre qui seraient autrement oubliés.

Finalement, lors d'une victoire de **B** où **W** est à court de libertés on a un polyomino avec ses trous complètement spécifié, sinon on applique récursivement l'algorithme sur les intersections encore libres. Après avoir énuméré tous les trous avec un certain $\mathbf{B}(0)$ et $\mathbf{W}(0)$, on place une pierre blanche sur $\mathbf{B}(0)$ et relance l'algorithme avec la position de départ suivante. La figure 3.9 illustre les différentes positions de départ utilisées pour générer les trous d'un polyomino.

3.3.3 Optimisations

Afin d'éliminer le plus rapidement possible un nombre maximal de parties où **B** perd, on conservera pour chaque intersection le nombre de pierres noires minimales requises pour revenir à l'origine. Le joueur noir pourra éviter de jouer sur des intersections inutilement "loin". À la figure 3.11 et à moins de mention contraire dans le reste de ce document, seules les intersections atteignables selon l'algorithme sont affichées.

On considère le graphe des coups possibles de **B**. Les sommets du graphe sont les posi-

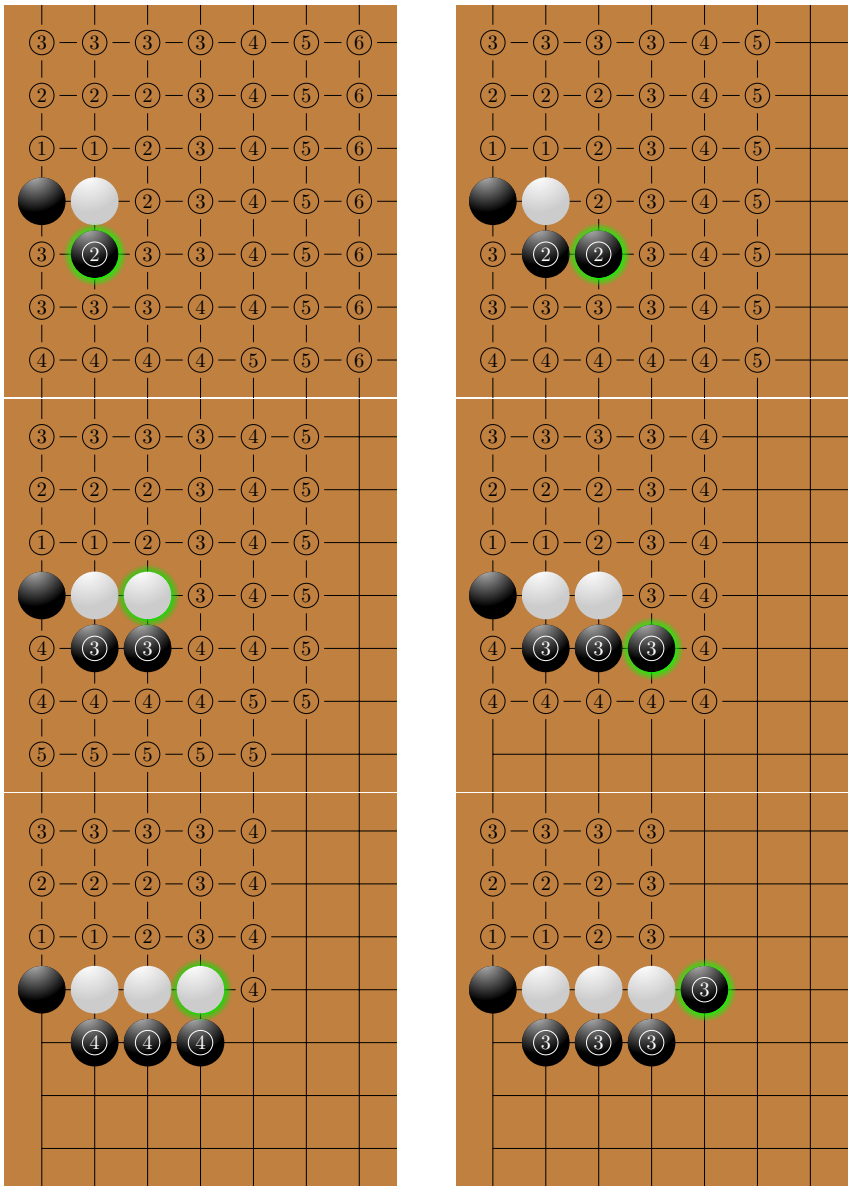


Figure 3.11: Nombre minimal de pierres noires requises

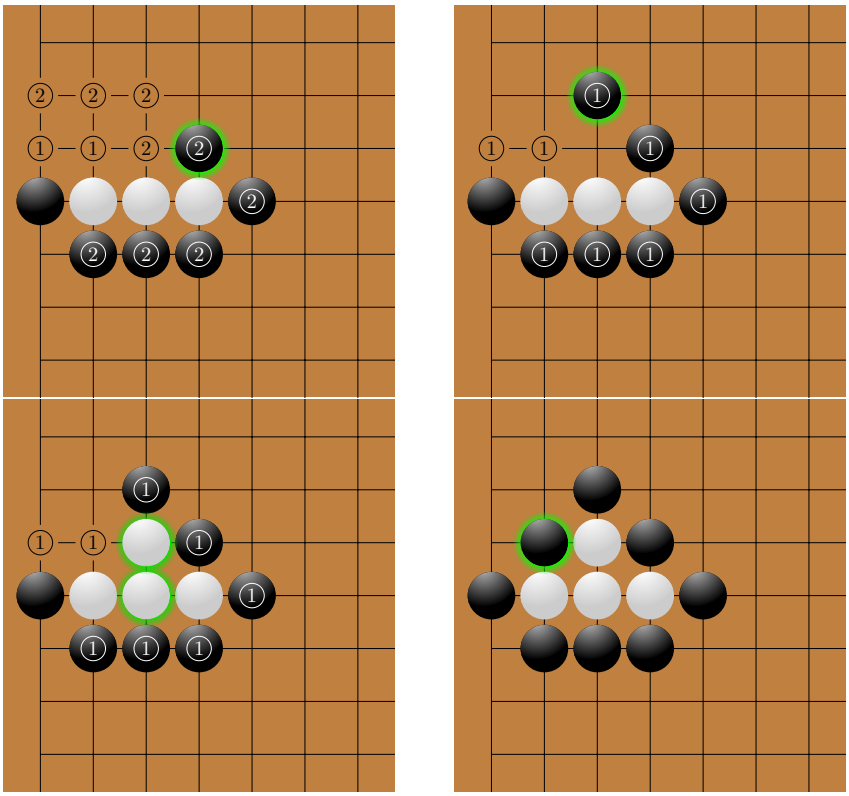


Figure 3.11: Nombre minimal de pierres noires requises

tions possibles, les arcs les coups de **B** et le poids des arcs le nombre de pierres noires qu'on ajoute en y jouant. On se rappelle que jouer sur une intersection libre ajoute une pierre noire tandis que jouer sur une pierre noire n'augmente pas le nombre de pierres en jeu. Nous utilisons ensuite l'algorithme de Dijkstra pour calculer le nombre de pierres noires minimales requises pour rejoindre l'origine de toutes les positions. L'utilisation d'une structure de données de type tas (heap en anglais) permet d'obtenir à la ligne 8 le tuple de distance minimale par l'opération *pop*.

Algorithme 5 Calcul des distances

Postconditions: Le tableau *distances* correctement mis à jour pour le prochain coup

```
 1: procedure CalculDesDistances
 2:     initialisation des distances à ∞
 3:     initialisation du tas H
 4:     marquer (0, 0) comme solutionné
 5:     ajouter (1, W + (0, 1)) à H
 6:     ajouter (1, W + (1, 1)) à H
 7:     tant que |H| > 0 faire
 8:         (d, pt) ← pop(H)
 9:         si pt n'est pas solutionné alors
10:             distances[pt] ← d
11:             pour tout  voisin de pt non-solutionné v faire
12:                 si v est noir alors
13:                     ajouter (d, v) à H
14:                 sinon si v est vide et d < nombre de pierres noires restantes alors
15:                     ajouter (d + 1, v) à H
16:             marquer pt comme solutionné
```

On notera finalement qu'à la ligne 14 on évite de traiter une intersection qui se trouve à une distance plus grande que le nombre de pierres noires restantes. Elle demeure, ainsi que toutes les autres intersections qui en dépendent pour leur chemin minimal, hors de portée pour le reste de la partie. La figure 3.11 illustre la progression du nombre de positions calculées par l'algorithme tout au long d'une partie.

3.3.4 Résultats expérimentaux

Les résultats expérimentaux présentés ici ont été obtenus en deux temps. Tout d'abord pour (Fortier *et al.*, 2013) nous avons développé l'algorithme principal des gominos. Les temps de calcul étant très long, plusieurs ordinateurs ont été utilisés en parallèle. Les résultats pour les polyominos de périmètre de site jusqu'à $n = 15$ ont ainsi été obtenus tels que rapportés au tableau 3.1.

Suite à une discussion avec Jérôme Fortier, j'ai remplacé les heuristiques de distances utilisées jusque là par l'algorithme de calcul présenté plus haut. On constate l'amélioration massive des performance sur le graphique 3.1.

n	Simplement 4-connexe	Simplement 8-connexe	8-trous permis
4	1	1	1
5	0	0	0
6	2	2	2
7	4	4	4
8	12	12	12
9	32	32	32
10	110	110	110
11	340	340	340
12	1193	1209	1209
13	4080	4256	4272
14	14786	15974	16166
15	53428	60232	61849

Tableau 3.1: Nombre de polyominos de périmètre de site n selon leur connectivité.

La taille de l'arbre des parties exploré par l'algorithme peut être borné grossièrement par 7^{2n+1} puisque **B** a accès à au plus 7 coups possibles, et ne peut pas jouer plus de $2n$ coups puisqu'il lui est impossible de se déplacer plus de 2 fois sur une intersection donnée (sinon **W** ne pourrait pas le rejoindre). Le nombre total de parties, tout comme le nombre de parties où **B** gagne, est donc borné par une courbe exponentielle. Il serait intéressant d'établir des bornes plus serrées.

En fait, il serait intéressant d'analyser plus en détail la relation entre le nombre de parties de gomino avec moins de n pierres noires et le nombres de parties où **B** gagne puisque le graphe à la figure 3.12 suggère une relation polynomiale entre le temps de calcul (autrement dit le nombre de parties examinées) et le nombre de polyominos (simplement 4-connexe en bleu, simplement 8-connexe en rouge). Plus précisément, pour un n donné avec $i = 0$ si on inclut les trous, $i = 1$ sinon, soit $\gamma_i(n)$ le total de parties de gomino de $\leq n$ pierres et $\beta_i(n)$ le nombre de parties où **B** gagne. Une régression linéaire sur ces données avec $R^2 = 0.9985$ suggère qu'on a

$$\gamma_0(n) \approx \beta_0(n)^{2.0376} \quad \text{et} \quad \gamma_1(n) \approx \beta_1(n)^{1.6665}. \tag{3.2}$$

Toutefois, les nouvelles données obtenues à l'aide de l'algorithme 5 tendent à montrer que la différence observée entre ces deux courbes était surtout due à l'exploration inutile de boucles intérieures. Les courbes marquées de $\times$ sont maintenant très proches l'une de l'autre.

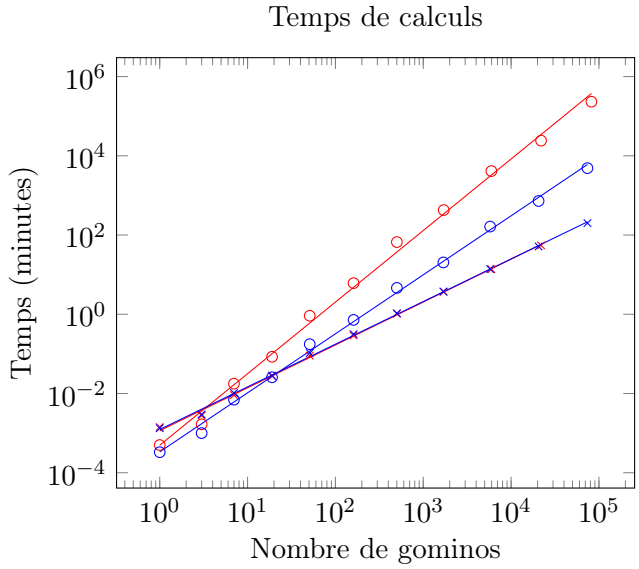


Figure 3.12: Temps de calcul des deux premières colonnes du tableau 3.1. (L'algorithme original est marqué par des $\circ$ tandis que l'algorithme optimisé est marqué par des $\times$)

CONCLUSION

Et voilà. Un chapitre de ma vie qui s'achève.

Le sujet est évidemment loin d'être épuisé. L'un des aspects auquel je me suis intéressé sans malheureusement avoir le temps d'en approfondir les détails est l'utilisation du mot de Lyndon associé à un mot de contour pour obtenir un point canonique sur le bord du polyomino plutôt que de confiner le polyomino dans le demi-plan pour obtenir le point W. Ceci imposerait des contraintes tout au long du mot et permettrait ainsi à chaque étape d'élaguer des branches de l'arbre de recherche. Par contre l'absence de contrainte dans le demi plan implique des changements majeurs dans le calcul du chemin le plus court, qu'on pourrait même devoir abandonner complètement. Il serait intéressant de voir l'effet net de ces deux facteurs sur le nombre de parties à examiner.

D'autre part, combiné avec l'utilisation de l'alphabet des premières différences, ceci nous permettrait d'énumérer les gominos à rotation près. En fait, on devrait pouvoir utiliser exactement le même algorithme pour générer les gominos fixes ou à rotation près simplement en utilisant des alphabets différents.

Du coté de l'algorithme de calcul des distances, quelques optimisations pourraient encore être implémentées. En effet, une fois les pierres noires atteintes il est inutile de calculer les valeurs précises des autres intersections : le prochain coup est soit sur une pierre noire (et donc n'ajoute pas de pierre), soit sur une intersection vide (et ajoute exactement une pierre). Ceci améliorerait les performances mais pas la classe de complexité de l'algorithme. Il serait intéressant de chercher un algorithme qui élimine encore plus rapidement les branches ne contenant pas de solutions. La nature incrémentale de la construction des chemins permet d'espérer un meilleur algorithme où on n'aurait pas à recalculer toutes les distances.

Le code source du programme de génération ainsi que celui de ce document et de ses figures est disponible sur Github à l'adresse https ://github.com/jerometremblay/memoire, en espérant que cela puisse éventuellement être utile à quelqu'un.

Merci de m'avoir lu jusqu'à la fin.

BIBLIOGRAPHIE

Avis, D. (1982). On the complexity of finding the convex hull of a set of points. *Discrete Applied Mathematics*, *4*(2), 81 – 86. http://dx.doi.org/http://dx.doi.org/10.1016/0166-218X(82)90065-8. Récupéré de http://www.sciencedirect.com/science/article/pii/0166218X82900658

Beauquier, D. et Nivat, M. (1991). On translating one polyomino to tile the plane. *Discrete Computational Geometry*, *6*, 575–592.

Blondin Massé, A. (2012). *À l'intersection de la combinatoire des mots et de la géométrie discrète : Palindromes, symétries et pavages.* (Thèse de doctorat). Université du Québec à Montréal.

Bousquet-Mélou, M. (1994). Codage des polyominos convexes et équations pour l'énumération suivant l'aire. *Discrete Applied Mathematics*, *48*(1), 21–43.

Bousquet-Mélou, M. (1996). A method for the enumeration of various classes of column-convex polygons. *Discrete Mathematics*, *154*(1-3), 1–25.

Bousquet-Mélou, M. (1998). New enumerative results on two-dimensional directed animals. *Discrete Mathematics*, *180*(1-3), 73–106.

Bousquet-Mélou, M. et Rechnitzer, A. (2003). The site perimeter of bargraphs. *Advances in Applied Mathematics*, *31*(1), 86–112.

Bresenham, J. E. (1965). Algorithm for computer control of a digital plotter. *IBM Syst. J.*, *4*(1), 25–30.

Brlek, S., Frosini, A., Rinaldi, S. et Vuillon, L. (2006). Tilings by translation : Enumeration by a rational language approach. *Electr. J. Comb.*, *13*(1).

Brlek, S., Koskas, M. et Provençal, X. (2011). A linear time and space algorithm for detecting path intersection in Z^d. *Theoretical Computer Science*, *412*(36), 4841–4850.

Brlek, S., Labelle, G. et Lacasse, A. (2008). On minimal moment of inertia polyominoes. In D. Coeurjolly, I. Sivignon, L. Tougne, et F. Dupont (dir.), *Discrete Geometry for Computer Imagery*, volume 4992 de *Lecture Notes in Computer Science* 299–309. Springer Berlin Heidelberg

Brlek, S., Lachaud, J.-O., Provençal, X. et Reutenauer, C. (2009). Lyndon+Christoffel = digitally convex. *Pattern Recognition*, *42*, 2239–2246.

Brlek, S., Tremblay, H., Tremblay, J. et Weber, R. (2014). Efficient computation of the outer hull of a discrete path. Dans *Discrete Geometry for Computer Imagery - 18th IAPR International Conference, DGCI 2014, Siena, Italy, September 10-12, 2014. Proceedings*, volume 8668 de *Lecture Notes in Computer Science*. Springer.

Chan, T. M. (1996). Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete & Computational Geometry*, *16*(4), 361–368.

Chaudhuri, B. et Rosenfeld, A. (1998). On the computation of the digital convex hull and circular hull of a digital region. *Pattern Recognition*, *31*(12), 2007 – 2016.

Christoffel, E. B. (1875). Observatio arithmetica. *Annali di Matematica*, *6*, 145–152.

Duval, J. P. (1983). Factorizing words over an ordered alphabet. *Journal of Algorithms*, *4*(4), 363 – 381.

Fortier, J., Goupil, A., Lortie, J. et Tremblay, J. (2013). Exhaustive generation of gominoes. *Theoretical Computer Science*, *502*, 76–87.

Freeman, H. (1961). On the encoding of arbitrary geometric configurations. *IRE Trans. Electronic Computer*, *10*, 260–268.

Fulton, W. (1997). *Young Tableaux : With Applications to Representation Theory and Geometry*. Cambridge University Press.

Gardner, M. (1958). Mathematical games. *Scientific American*, Sept. 182–192, Nov. 136–142.

Golomb, S. W. (1954). Checker boards and polyominoes. *Amer. Math. Monthly*, *61*, 675–682.

Golomb, S. W. (1996). *Polyominoes : Puzzles, Patterns, Problems, and Packings*. Princeton : Princeton Academic Press.

Goupil, A., Cloutier, H. et Nouboud, F. (2010). Enumeration of polyominoes inscribed in a rectangle. *Discrete Applied Mathematics*, *158*(18), 2014–2023.

Graham, R. (1972). An efficient algorith for determining the convex hull of a finite planar set. *Information Processing Letters*, *1*(4), 132 – 133.

Grosberg, A. Y. (2014). Annealed lattice animal model and flory theory for the melt of non-concatenated rings : towards the physics of crumpling. *Soft Matter*, *10*, 560–565.

Gross, J. et Tucker, T. (1987). *Topological Graph Theory*. Dover Books on Mathematics Series. Dover Publications.

Gruber, P. M. (2007). *Convex and discrete geometry*. Berlin : Springer.

Jensen, I. (2001). Enumerations of lattice animals and trees. *J. Stat. Mechanics*, *102*, 865–881.

Jensen, I. et Guttmann, A. J. (2000). Statistics of lattice animals (polyominoes) and polygons. *Journal of Physics*, *33*, L257–L263.

Kim, M., Lee, E., Cho, H. et Park, K. (1997). A visualization technique for dna walk plot using k-convex hull. Dans *Proceedings of the Fifth International Conference in Central Europe in Computer Graphics and Visualization*, 212–221.

Kim, Y. (1992). Recognition of form features using convex decomposition. *Computer-Aided Design*, *24*(9), 461 – 476.

Klarner, D. A. et Rivest, R. L. (1972). *A procedure for improving the upper bound for the number of n-ominoes.* Rapport technique, Stanford, CA, USA.

Kraaikamp, C. et Meester, R. (1995). Ergodic properties of a dynamical system arising from percolation theory. *Ergodic Theory and Dynamical Systems*, *15*(4), 653–661.

Lubensky, T. et Isaacson, J. (1979). Statistics of lattice animals and dilute branched polymers. *Physical Review A*, *20*(5).

McCallum, D. et Avis, D. (1979). A linear algorithm for finding the convex hull of a simple polygon. *Information Processing Letters*, *9*(5), 201–206.

Melkman, A. A. (1987). On-line construction of the convex hull of a simple polyline. *Inform. Process. Lett.*, *25*(1), 11–12.

Pólya, G. (1969). On the number of certain lattice polygons. *J. Combinatorial Theory*, *6*, 102–105.

Provençal, X. (2008). *Combinatoire des mots, géométrie discrète et pavages.* (Thèse de doctorat). D1715, Université du Québec à Montréal.

Redelmeier, D. H. (1981). Counting polyominoes : yet another attack. *Discrete Mathematics*, *36*, 191–203.

Rosenfeld, A. (1969). Picture processing by computer. *ACM Comput. Surv.*, *1*(3), 147–176.

Sherali, H. D. et Adams, W. P. (1994). A hierarchy of relaxations and convex hull characterizations for mixed-integer zero—one programming problems. *Discrete Applied Mathematics*, *52*(1), 83 – 106.

Sieben, N. (2008). Polyominoes with minimum site-perimeter and full set achievement games. *European Journal of Combinatorics*, *29*(1), 108–117.

Sloane, N. J. A. (2005). The On-Line Encyclopedia of Integer Sequences. `http://oeis.org/A001168`. Number of fixed polyominoes with n cells.

Sloane, N. J. A. (2011). The on-line encyclopedia of integer sequences. published electronically at http ://www.research.att.com/ ?njas/sequences.

Spitzer, F. (1956). A combinatorial lemma and its application to probability theory. *Transactions of the American Mathematical Society*, *82*(2), pp. 323–339.

Stanley, R. P. (1999). *Enumerative Combinatorics*. Cambridge : Cambridge University Press.

van Emde Boas, P. (1980). On the omega(n log n) lower bound for convex hull and maximal vector determination. *Inf. Process. Lett.*, *10*(3), 132–136. `http://dx.doi.org/10.1016/0020-0190(80)90064-2`. Récupéré de `http://dx.doi.org/10.1016/0020-0190(80)90064-2`

Viennot, X. G. (1992). A survey of polyomino enumeration. Dans P. Leroux et C. Reutenauer (dir.). *Séries formelles et combinatoires algébrique, 4th International Conference, Montréal, June 15-19, 1992*, volume 11 de *Publications du LACIM*, 399–420. Springer.