UNIVERSITÉ DU QUÉBEC

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

COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN MATHÉMATIQUES ET INFORMATIQUE

PAR
RAPHAËL L'HEUREUX

RECHERCHE DES SERPENTS ET DES FORÊTS DE SERPENTS MAXIMAUX DANS LE PLAN
APPROCHES EXHAUSTIVES ET APPRENTISSAGE MACHINE

DÉCEMBRE 2022

Résumé

Ce mémoire porte sur les polyominos serpents d'aire maximale ainsi que sur leurs forêts. Il s'agit de l'introduction des forêts de serpents dans la littérature scientifique. Deux algorithmes d'énumération de ces forêts de serpents sont présentés, dont une adaptation de la méthode des matrices de transfert qui produit un graphe pour chaque largeur du rectangle pouvant être utilisé pour l'énumération, la production d'une série génératrice ou pour l'apprentissage machine. Ce projet contient aussi la première tentative de génération des serpents d'aires maximales grâce à l'apprentissage machine.

Abstract

This master's thesis discusses maximal area snake polyominoes and their forests. We introduce here for the first time the concept of forest of snakes. Two enumeration algorithms are presented, one of which is an adaptation of the transfer matrix method which produces a graph that depends on the width of the rectangle that can be used to enumerate snake forests with respect to different parameters such as the length, the height or the number of connected components and to calculate the corresponding generating function or for machine learning. This project is also, to our knowledge, the first attempt at maximal area snake generation using machine learning.

Merci à mes directeurs de recherche, Alain Goupil et Usef Faghihi, pour les nombreuses heures de discussions enrichissantes. Votre encadrement a été exemplaire.

Merci à Fadel Touré de m'avoir permis d'utiliser son serveur pour effectuer les entraînements des modèles de réseaux neuronaux.

Merci à la fondation de l'UQTR ainsi qu'à Alain Goupil pour les bourses d'études.

Merci à l'équipe de badminton des Patriotes de m'avoir donné une raison supplémentaire de poursuivre mes études. Nous sommes Olivier Nadeau.

Merci à ma famille pour le support et les encouragements. Je vous aime.

Table des matières

Liste des tableaux

Table des figures

Chapitre 1

Introduction

Depuis quelques années, les efforts de recherches dans le domaine de l'intelligence artificielle se multiplient. Ces efforts ont amené une évolution rapide des applications de l'apprentissage machine. En effet, l'intelligence artificielle est désormais présente dans la majorité des sphères de la vie humaine, que ce soit dans le domaine du logiciel, des transports ou de la santé par exemple.

Toutefois, malgré l'importance des mathématiques dans le développement de cette technologie, peu de travaux ont pour objectif d'utiliser l'intelligence artificielle dans le but d'approfondir les connaissances en mathématiques.

Dans ce cas-ci, il s'agit du domaine des mathématiques combinatoires. L'intérêt principal des mathématiques combinatoires est la recherche des différentes façons d'assembler les éléments d'un ensemble fini tout en respectant diverses règles. L'une des catégories de problèmes combinatoires est celle des polyominos, qui sont des ensembles de cellules connectées par les côtés. Parmi ces polyominos, il existe les polyominos serpent dont l'énumération forme le sujet principal de cette étude.

Le premier objectif de ce projet est la description et l'implémentation d'algorithmes de génération exhaustive des serpents maximaux et des forêts de serpents maximales, principalement l'adaptation de la méthode des matrices de transfert pour la génération de ces objets. L'adaptation de cette méthode permettrait de produire des automates de génération des forêts de serpents qui pourront servir au développement de diverses stratégies d'apprentissage automatique.

Le deuxième objectif est d'utiliser l'apprentissage machine afin de générer les serpents maximaux de manière automatique. Ce projet constitue, à notre connaissance, la première tentative d'application de l'intelligence artificielle à un problème de type chemin auto-évitant. L'utilisation de l'intelligence artificielle dans ce projet a pour intérêt principal le développement d'un nouvel outil d'exploration des serpents maximaux pour contrer les limitations dues à la complexité exponentielle des algorithmes de recherche exhaustive. Un tel outil permettrait aux mathématiciens d'approfondir leur capacité exploratoire afin d'effectuer plus d'observations, ce qui est un atout significatif dans la découverte et la confirmation de nouvelles conjectures.

1.1 Structure du mémoire

Le chapitre 2 présentera les définitions importantes ainsi que les concepts de base de ce projet. Ces concepts sont les polyominos, les serpents, les forêts de serpents et la différence entre un serpent (ou une forêt de serpents) à symétrie près et à translation près. Il s'agira aussi de la première apparition du concept de forêt de serpents dans la littérature scientifique.

Le chapitre 3 portera sur la génération exhaustive des serpents et des forêts de serpents. Deux méthodes y seront discutées. La première est la méthode de la fouille en profondeur fonctionnant cellule par cellule. La deuxième est une adaptation de la

méthode des matrices de transfert permettant de générer ligne par ligne les serpents et les forêts de serpents du rectangle d'inscription.

Le chapitre 4 introduira la première tentative d'application de l'apprentissage machine pour la génération des serpents maximaux dans le plan. Plusieurs configurations de réseaux antagonistes génératifs et l'impact de leurs différents paramètres sur la génération des serpents seront analysés.

Le chapitre 5 synthétisera les résultats et les éléments importants de ce projet. Il laissera place au chapitre 6, qui conclura ce mémoire en faisant un retour sur les objectifs et en présentant certaines perspectives de travaux futurs.

Chapitre 2

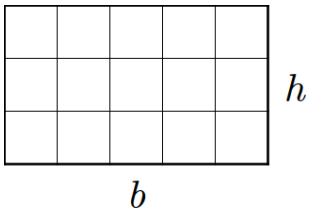
Définition des concepts

2.1 Définitions

Avant d'élaborer davantage le sujet, il est essentiel de se munir de certaines définitions et notations :

- b **et** h : Dimensions de la base et de la hauteur d'un rectangle $b \times h$. Ils sont représentés dans la figure 2.1 ;
- **Cellule** : Un carré 1×1 dans un rectangle ;
- **Cellule sélectionnée** : Une cellule sélectionnée est une cellule qui fait partie d'un polyomino ;
- **Cellule vide (non sélectionnée)** : Une cellule vide est une cellule qui ne fait pas partie d'un polyomino ;
- **Ensemble connexe de cellules** : Un ensemble connexe de cellules est un ensemble de cellules sélectionnées connectées entre elles par les côtés.
- **Degré d'une cellule** : Le degré d'une cellule correspond au nombre de cellules sélectionnées étant connectées par un côté. Le degré d'une cellule peut donc être entre 0 et 4 inclusivement.

FIGURE 2.1 – Emplacements des dimensions des rectangles dans les figures de ce document



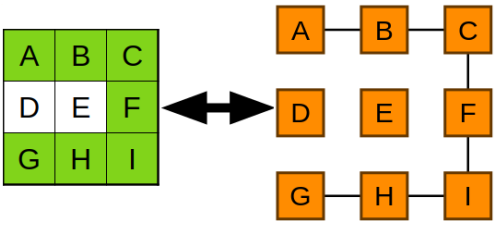
2.2 Polyominos

Un polyomino est un ensemble connexe de cellules dans le plan. Les polyominos peuvent être séparés en plusieurs classes telles que les polyominos convexes, les polyominos arbres et les polyominos serpents par exemple. L'énumération des différents polyominos est un problème combinatoire complexe ayant fait l'objet de plusieurs travaux, tels que ceux de Redelmeier [1] et ceux de Jensen [2][3], visant à améliorer leurs algorithmes d'énumération. Certaines classes de polyominos telles que celles des polyominos convexes et des polyominos Manhattan ont été énumérées avec succès. Il existe aussi des fonctions génératrices pour certaines classes spécifiques [4][5]. Toutefois, plusieurs classes de polyominos telles que les polyominos à translation près, les polyominos serpents selon l'aire et les polyominos arbres ne possèdent toujours pas de solution au problème de leur énumération.

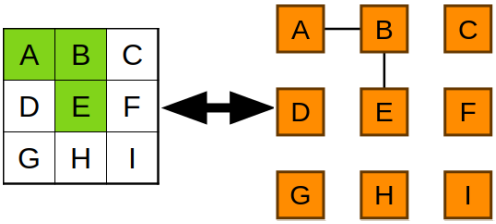
Les polyominos peuvent contenir des cycles. Un polyomino contient un cycle lorsqu'il est possible de parcourir plus d'une fois la même cellule sélectionnée lors d'un parcours dirigé dans ce polyomino. Autrement dit, le concept de cycle dans un polyomino est similaire à celui de la théorie des graphes. Comme le montre la figure 2.2, un polyomino peut être représenté par un graphe où chaque cellule est un sommet et où les cellules adjacentes sont liées entre elles par des arêtes. Ainsi, le polyomino contient un cycle s'il existe, dans le graphe correspondant, une chaîne simple où les deux extrémités sont le même sommet.

FIGURE 2.2 – Exemples de polyominos avec et sans cycle accompagnés du graphe correspondant

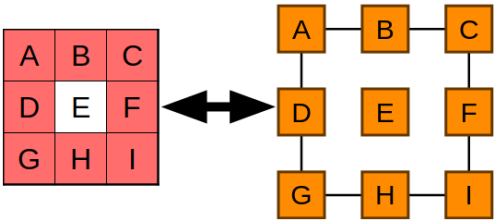
(a) Polyominos sans cycle



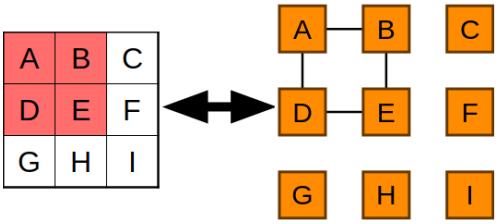
(b) Polyominos sans cycle



(c) Polyominos avec cycle



(d) Polyominos avec cycle



D'autres concepts importants en lien avec les polyomminos sont l'aire d'un polyomino ainsi que son rectangle d'inscription. L'aire d'un polyomino correspond au nombre de cellules sélectionnées qu'il contient. Le rectangle d'inscription d'un polyomino est le plus petit rectangle pouvant contenir ce polyomino.

Ce projet s'intéresse plus précisément aux tailles maximales des polyominos serpents et de leurs forêts contenus dans un rectangle $b \times h$ donné.

2.3 Serpents et forêts de serpents

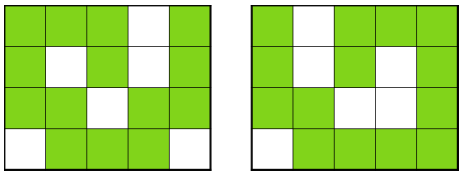
2.3.1 Serpents

Un serpent est un polyomino sans cycle ne pouvant contenir que des cellules de degré inférieur ou égal à 2. Cela signifie aussi qu'un serpent peut alors être composé d'une cellule isolée. Un serpent est considéré maximal s'il est impossible de construire

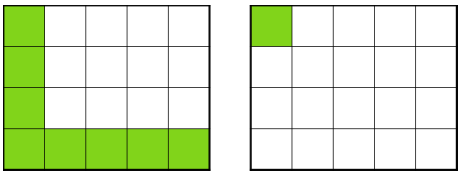
un serpent possédant plus de cellules sélectionnées dans son rectangle d'inscription. La figure 2.3 présente des exemples de serpents maximaux, non maximaux ainsi que des exemples des structures qui ne sont pas des serpents.

FIGURE 2.3 – Exemples de serpents maximaux, non maximaux et de structures n'étant pas des serpents

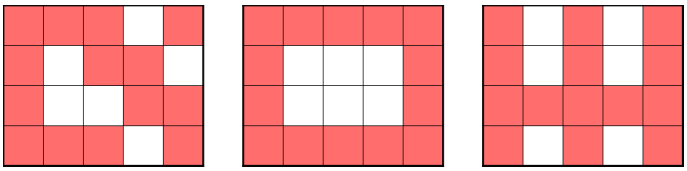
(a) Serpents maximaux



(b) Serpents non maximaux



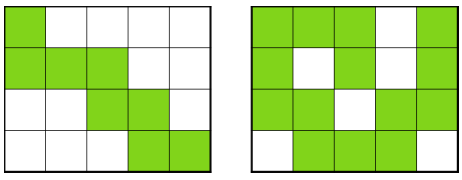
(c) Polyominos n'étant pas des serpents (plus d'un ensemble connexe de cellule, présence d'un cycle ou plus de deux extrémités)



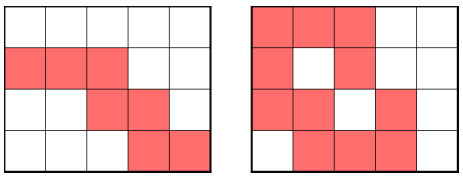
Un serpent est inscrit dans un rectangle s'il possède au moins une cellule sélectionnée sur chaque côté du rectangle. La figure 2.4 montre des exemples de serpents inscrits ainsi que de serpents non inscrits.

FIGURE 2.4 – Exemples de serpents inscrits et non inscrits

(a) Serpents inscrits



(b) Serpents non inscrits



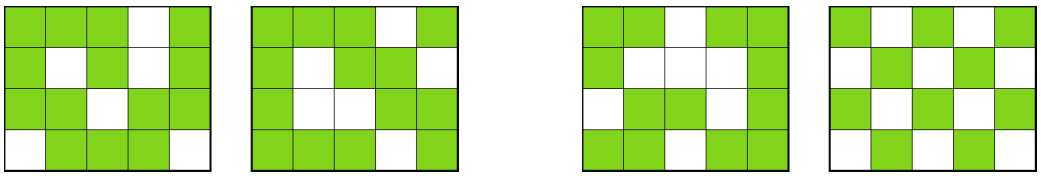
2.3.2 Forêts de serpents

Ce projet effectue l'introduction des forêts de serpents dans la littérature scientifique. Une forêt de serpents est un ensemble de serpents pouvant contenir un serpent

ou plus. Ainsi, tous les serpents sont considérés comme des forêts de serpents mais ce ne sont pas toutes les forêts qui sont des serpents. Une forêt de serpents est considérée maximale lorsqu'il est impossible de construire une forêt possédant plus de cellules sélectionnées dans le rectangle auquel elle appartient. La figure 2.5 présente des exemples de forêts de serpents maximales et non maximales.

FIGURE 2.5 – Exemples de forêts de serpents maximales et non maximales

(a) Forêts de serpents maximales (b) Forêts de serpents non maximales

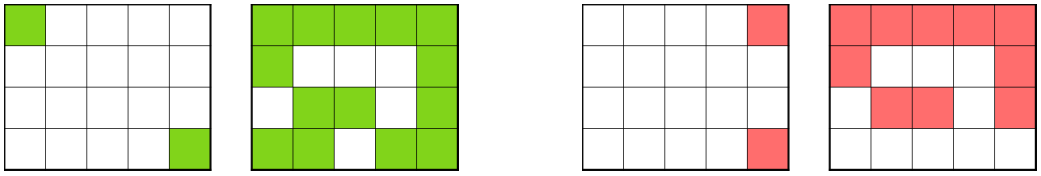


Les forêts de serpents sont introduites dans le but principal d'apporter un outil supplémentaire pour l'étude des serpents. En effet, les sous-rectangles d'un serpent inscrit contiennent un ensemble de cellules ne formant pas toujours un serpent mais formant toujours une forêt de serpents. Alors il est nécessaire d'investiguer les forêts de serpents afin de pouvoir investiguer les serpents en s'appuyant sur les sous-rectangles du rectangle d'inscription. De plus, il est possible de retirer n'importe laquelle des cellules d'une forêt de serpents et la structure résultante sera encore une forêt de serpents, ce qui n'est pas le cas pour les serpents et qui facilite le raisonnement inductif.

Comme les serpents, une forêt de serpents est inscrite dans un rectangle si elle possède au moins une cellule sélectionnée sur chaque côté du rectangle. La figure 2.6 montre des exemples de forêts de serpents inscrites ainsi que de forêts non inscrites.

FIGURE 2.6 – Exemples de forêts de serpents inscrites et non inscrites

(a) Forêts de serpents inscrites (b) Forêt de serpents non inscrites



2.3.3 Symétries et translations

Il est aussi important d'établir la distinction entre des serpents (ou des forêts de serpents) à translation près et à symétrie près. Les serpents à translation près sont aussi appelés les serpents fixes et les serpents à symétrie près sont aussi appelés les serpents libres. Les concept de serpent ou de forêt de serpents à translation près ou à symétrie près peuvent être définis en termes de relation d'équivalence selon leur positionnement dans le plan discret.

Tout d'abord, deux serpents (ou forêt de serpents) $S1$ et $S2$ dans le plan discret sont dits équivalents à translation près si l'un s'obtient de l'autre par une translation. On écrit alors $S1 \sim_T S2$. La relation $\sim_T$ entre les serpents est une relation d'équivalence et un serpent à translation près est un représentant d'une classe d'équivalence de la relation $\sim_T$. Pour une classe d'équivalence de serpents à translation près donnée, on peut fixer le représentant en choisissant celui dont le rectangle circonscrit a son coin inférieur gauche à l'origine du plan cartésien. De cette façon, on se libère de la contrainte de situer les serpents dans le plan cartésien et on peut parler sans ambiguïté du rectangle circonscrit d'un serpent.

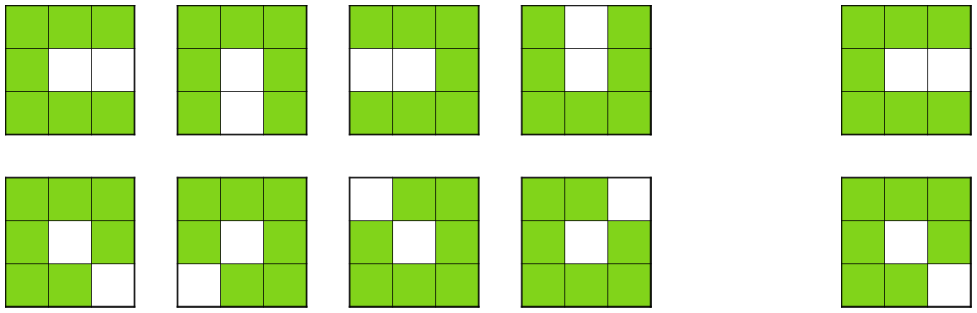
Ensuite, deux serpents (ou forêts de serpents) $S1$ et $S2$ sont équivalents à symétrie près si l'un s'obtient de l'autre par une symétrie du plan (rotation ou réflexion). On écrit alors $S1 \sim_S S2$. La relation $\sim_S$ entre les serpents est une relation d'équivalence et un serpent à symétrie près est un représentant d'une classe d'équivalence de la relation $\sim_S$. Un polyomino à symétrie près est un représentant d'une classe d'équivalence pour la symétrie.

En d'autres mots, lorsque l'on parle des serpents (ou forêts de serpents) inscrits, l'ensemble des serpents à translation près d'un rectangle contient toutes les possibilités de serpents du rectangle alors que l'ensemble de serpents à symétrie près n'inclut qu'un seul représentant de la classe d'équivalence de la relation $\sim_S$. Il y a donc un maximum

de quatre possibilités de serpents fixes pour chaque serpent libre, bien qu'en fonction de sa structure, un serpent libre peut aussi correspondre uniquement à un ou deux serpents fixes. Ce qui est aussi vrai pour les forêts de serpents. Ainsi, l'ensemble de serpents ou de forêts de serpents à symétrie près offre les avantages d'être plus petit et donc plus facile à observer. L'ensemble des serpents ou des forêts de serpents à translation près est cependant plus complet et donc à privilégier pour l'analyse des patrons. La figure 2.7 présente, en exemple, les 8 serpents maximaux du carré 3×3 à translation près et les 2 serpents maximaux à symétrie près.

FIGURE 2.7 – Ensembles des serpents maximaux du carré 3×3

(a) À translation près (tous les serpents maximaux) (b) À symétrie près (un réprésentant par classe d'équivalence de $\sim_S$)



Le tableau 2.1 contient les suites correspondantes aux nombres de serpents inscrits libres et fixes pour les rectangles de largeur 2, 3 et 4 en fonction de la hauteur (h) du rectangle. Ces nombres ont été obtenus à partir de la méthode d'énumération des serpents utilisant la fouille en profondeur présentée à la sous-section 3.2.1 et légèrement modifiée pour conserver tous les serpents inscrits et pas uniquement les serpents maximaux. Il est intéressant de constater que la suite du nombre de serpents inscrits fixes dans les rectangles $2 \times h$ fait partie de la base de données de « The Online Encyclopedia of Integer Sequences » (OEIS) [6] et correspond à la suite du nombre d'appels récursifs nécessaires pour calculer le n^{e} nombre de Fibonacci pour $h = n + 2$. Il s'agit de la seule des suites du tableau à faire partie de l'encyclopédie OEIS.

TABLE 2.1 – Suites du nombre de serpents inscrits libres et fixes des rectangles de largeur 2, 3 et 4

	2xh		**3xh**		**4xh**	
h	**Libres**	**Fixes**	**Libres**	**Fixes**	**Libres**	**Fixes**
2	1	4				
3	3	8	9	28		
4	4	14	21	76	82	324
5	8	24	51	184	295	1148
6	11	40	111	422	904	3600
7	20	66	249	944	2710	10716
8	29	108	532	2074	7750	30936
9	50	176	1155	4498	21812	86836
10	75	286	2449	9672	59521	237856
11	126	464	5239	20682		639908
12	194	752	11083	44056		1699950
13	321	1218	23548	93588		
14	503	1972	49753	198406		
15	825	3192		419988		
16	1308	5166		888038		
17	2134	8360		1876114		
18	3409	13528				
19	5544	21890				
20	8899	35420				
21	14444	57312				
22	23255	92734				
23	37700	150048				
24	60812	242784				
25	98513	392834				

Chapitre 3

Énumération des serpents et des forêts de serpents d'aire maximale

3.1 Revue de littérature

Ce mémoire porte sur l'étude des serpents maximaux et des forêts de serpents maximales. Ces structures spécifiques ont été créées dans le cadre d'une étude plus large, menée par le professeur Alain Goupil, sur les polyominos serpents et les polycubes arbres. L'étude des serpents maximaux est importante puisqu'elle pourrait permettre d'améliorer la performance des algorithmes de génération exhaustive de cette famille de polyominos. Les serpents maximaux ainsi que les forêts de serpents maximales n'ont pas été étudiés auparavant et ne font pas partie de la littérature. Par contre, les polyominos ont été investigués du point de vue combinatoire, algorithmique ainsi que de celui de la physique statistique pour, par exemple, le phénomène de percolation.

En effectuant cette revue de littérature, il a été possible de constater qu'un nombre

relativement faible de publications sur les polyominos serpents et leurs forêts font partie de la littérature. Certains travaux comme celui de Goupil et al. [7] visant à trouver des fonctions génératrices pour les serpents partiellement dirigés ont permis d'approfondir la compréhension de cette classe de serpents. Par contre, le problème actuel se situe plutôt dans une autre catégorie puisque les serpents de la présente étude ne sont ni dirigés ni prudents [8] et doivent être maximaux ou faire partie d'une forêt de taille maximale. Il existe aussi certaines méthodes permettant de compter les serpents d'un rectangle telles que celle présentée par Barrientos et Minion [9]. Leur méthode se base sur deux familles de serpents, soit les serpents en escalier et les serpents de hauteur 2, et sur un système de partitionnement rendu possible par la structure précise de ces familles de serpents. Pour ce qui est de l'énumération des serpents, Hugo Lessard présentait en 2013 son programme de génération des serpents par colonne lors d'un projet de fin d'études à l'UQTR [10].

Toutefois, il est possible d'élargir le champs de la recherche et de se rapporter aux très nombreuses publications touchant les problèmes des chemins auto-évitants (CAE) [11]. En effet, le problème des serpents et des forêts de serpents dans le plan constitue une forme de CAE, puisque ce sont des chemins auto-évitants épais. La différence majeure entre le problème actuel et les CAE traditionnels est que ces dernières se déplacent de sommet en sommet sur la grille discrète plutôt que de cellule en cellule. Une autre différence entre les serpents et les CAE est que, du point de vue de la théorie des graphes, les serpents ont la propriété d'être des sous-graphes induits. Cette distinction permet de faire le lien entre les serpents de la présente étude avec le problème du serpent dans la boîte dans l'hypercube B_n [12][13][14] et, plus généralement, de la génération de serpents dans un graphe quelconque.

Dans le domaine des CAE, la méthode des matrices de transfert a été adaptée par Doron Zeilberger [15][16][17] afin de compter les différentes possibilités et d'extraire certaines statistiques. La méthode des matrices de transfert a aussi été appliquée à une famille différente de polyominos par Marie-Ève Pellerin [18].

3.2 Méthode de la fouille en profondeur (cellule par cellule)

3.2.1 Énumération des serpents maximaux

La méthode de recherche cellule par cellule est la plus simple et la plus basique. Son but est simplement de générer exhaustivement toutes les possibilités de serpents du rectangle. Cela est fait en ajoutant une par une les cellules à la tête du serpent courant jusqu'à ce qu'il soit impossible d'en ajouter une nouvelle. Cette méthode utilise donc un algorithme récursif de fouille en profondeur [19] parcourant tous les serpents possibles à partir de chaque cellule du rectangle vide.

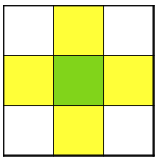
Il est aussi possible d'accélérer la recherche en ne l'effectuant qu'à partir des cellules d'un seul quartier du rectangle vide, mais cela ne permettra pas d'avoir tous les serpents maximaux à translation près à moins d'appliquer les symétries sur tous les serpents générés et de retirer les doublons produits.

Ainsi, à chaque cellule, une vérification est effectuée pour s'assurer qu'il est possible de continuer le serpent en sélectionnant une autre cellule. S'il est impossible de continuer de sélectionner une cellule, l'ensemble de serpents trouvé est retourné à l'étape de récursivité précédente et la fouille se poursuit à partir d'une autre cellule. Pour sélectionner une nouvelle cellule et continuer la fouille, son éligibilité est basée sur les critères suivants :

— La nouvelle cellule est adjacente à la cellule de tête du serpent (en vert sur la figure 3.1), elle est choisie parmi une des cellules jaunes sur la figure 3.1 ;
— La nouvelle cellule existe et n'est pas hors du rectangle ;
— La nouvelle cellule n'est pas déjà sélectionnée ;
— La nouvelle cellule ne possède actuellement qu'un seul voisin immédiat sélec-

tionné (soit la cellule verte sur la figure 3.1).

FIGURE 3.1 – Cellules fouillées (en jaunes) à partir de la cellule de tête (en vert) à chaque étape récursive de la fouille en profondeur pour l'énumération des serpents



Lorsque toutes les nouvelles cellules éligibles à partir de la cellule de tête courante ont été fouillées, les différents ensembles de serpents trouvés sont comparés et tous ceux qui contiennent le plus grand nombre de cellules sélectionnées sont retournés à l'étape précédente. L'exploration se poursuit ainsi jusqu'à ce que toutes les possibilités aient été parcourues à l'intérieur d'un rectangle donné.

L'annexe A.1 présente l'algorithme complet de recherche des serpents maximaux en pseudocode.

3.2.2 Énumération des forêts de serpents maximales

Pour ce qui est de l'énumération des forêts de serpents, il pourrait être intuitif de tenter d'adapter l'algorithme d'énumération des serpents décrit précédemment en étendant les critères de continuation de la recherche afin de prendre en compte la possibilité de création de nouveaux serpents à proximité de la tête du serpent actuel. Par exemple, pour sélectionner une cellule et continuer la fouille à partir de cette cellule, son éligibilité serait basée sur les critères suivants :

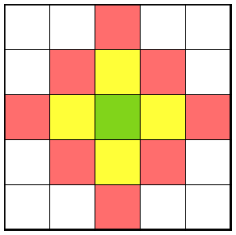
— Pour une cellule adjacente à la tête du serpent en construction (en jaune sur la figure 3.2) :
  — La cellule existe et n'est pas hors du rectangle ;
  — La cellule n'est pas déjà sélectionnée ;

- La cellule ne possède actuellement qu'un seul voisin immédiat sélectionné (soit la cellule de tête courante).

— Pour une cellule voisine en diagonale (une cellule connectée par un sommet) ou pour un deuxième voisin horizontal ou vertical de la tête du serpent en construction (en rouge sur la figure 3.2) :

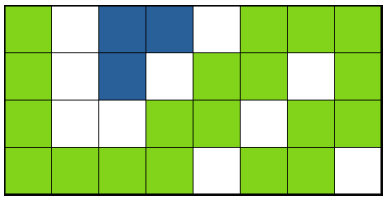
- La cellule existe et n'est pas hors du rectangle ;
- La cellule n'est pas déjà sélectionnée ;
- La cellule ne possède aucun voisin sélectionné.

FIGURE 3.2 – Cellules fouillées (en jaunes et en rouges) à partir de la cellule de tête (en vert) à chaque étape récursive de la fouille en profondeur tentant d'énumérer les forêts de serpents maximales



Toutefois, ces critères supplémentaires ne permettent pas d'énumérer toutes les forêts de serpents maximales. Par exemble, la figure 3.3 montre une forêt de serpents maximale du rectangle 4×8 impossible à atteindre puisque ces deux serpents ne possèdent aucune extrémité située dans la zone de recherche d'une cellule d'extrémité de l'autre serpent.

FIGURE 3.3 – Forêt de serpents maximale du rectangle 8×4 possédant au moins un serpent dont les deux extrémités ne sont pas dans la zone de recherche d'une extrémité d'un autre serpent



Afin de tenter d'obtenir toutes les forêts de serpents maximales, il faudrait vérifier le rectangle complet afin de s'assurer qu'il est impossible d'y ajouter une nouvelle

cellule à chaque fois que la tête du serpent en cours de construction ne possède plus de cellule éligible. Une cellule peut être ajoutée à la forêt si elle a un degré inférieur ou égal à 2, que ses voisines immédiates sélectionnées ont toutes un degré inférieur à 2 et que son ajout ne crée pas de cycle. S'il existe des cellules pouvant être ajoutées, la fouille en profondeur doit être continuée à partir de chacune de ces cellules.

Ensuite, il faudrait encore démontrer que cela permet bien de générer toutes les forêts de serpents maximales. Donc, en plus de ne pas être si simple d'implémentation, cette méthode d'énumération des forêts de serpents maximales ne permet peut-être pas l'énumération complète des forêts maximales. Elle est donc fortement déconseillée.

De façon alternative, il est préférable d'utiliser un algorithme d'énumération binaire en effectuant un élagage lorsque la sélection d'une cellule enfreint les règles structurelles d'une forêt de serpents. Cette méthode est plus rapide à implémenter et offre aussi l'avantage d'énumérer toutes les forêts de serpents d'un rectangle incluant les forêts non-maximales. Elles peuvent ensuite être traitées afin de conserver uniquement les forêts de serpents inscrites ou les forêts de serpents maximales. L'algorithme 1 présente cette adaptation en pseudocode.

3.3 Méthode de la matrice de transfert (ligne par ligne)

Cette méthode utilise la construction d'un graphe orienté, ou plus précisément d'un automate déterministe, associé aux rectangles d'une largeur b donnée. Les noeuds de ce graphe sont les différentes possibilités de lignes de longueur b du rectangle en tenant compte de la ligne précédente. Chaque arête (L_1, L_2) du graphe est pondérée par un monôme mesurant l'impact de l'ajout de la ligne L_2 sur l'aire et le nombre de composantes connexes de la forêt de serpents résultante en fonction de la ligne L_1.

Algorithm 1 Énumération des forêts de serpents

Require: $rectangle_{b \times h}$
1: $n = 0$
2: $aireRect \leftarrow b * h$
3: $forets \leftarrow$ empty list
4: **procedure** ENUMERERFORETS($rect, n$)
5: **if** $n = aireRect$ **then**
6: $forets.ajouter(rect)$
7: **return**
8: $i \leftarrow n \div b$ ▷ Assigne la partie entière seulement
9: $j \leftarrow n \mod b$
10: $rect[i, j] \leftarrow 1$
11: **if** DEGREMAXCELLULESVOISINES(i, j) $\leq$ 2 **and** FORMEUNCYCLE(i, j) is **false then** ▷ Utilise lièvre et tortue décrit dans la section 3.3.2
12: ENUMERERFORETS($copie(rect), n + 1$)
13: $rect[i, j] \leftarrow 0$
14: ENUMERERFORETS($copie(rect), n + 1$)
15: **return**

Cette section présente la construction de matrices d'adjacences pour les forêts de serpents. Elle peut cependant aussi être utilisée et même optimisée pour être spécifique aux serpents.

3.3.1 Définitions

Avant de débuter cette section, il est important de préciser que les explications et les illustrations utilisent comme convention que les lignes du rectangle sont ajoutés de bas en haut. De plus, certains termes utilisés doivent être définis :

— **Noeud** : Un noeud du graphe orienté est une ligne de cellules de longueur b ayant des caractéristiques uniques. Ces caractéristiques sont décrites dans le paragraphe suivant ;

— **Segment** : Un segment est une suite de cellules consécutives sélectionnées appartenant à la même ligne ;

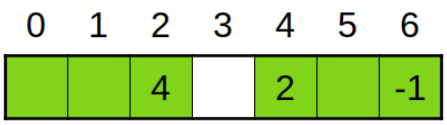
— **Indice de colonne** : L'indice de colonne est le numéro de la colonne dans le

rectangle. Les colonnes sont numérotés de gauche à droite en commençant par 0 ;

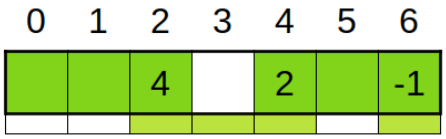
— **Étiquette de liaison** : L'étiquette de liaison est une façon de noter la présence de liens « souterrains » entre les segments de la ligne. Seules les cellules sélectionnées étant l'extrémité d'un segment peuvent posséder une étiquette de liaison. Cette étiquette est l'indice de colonne de la cellule d'extrémité d'un autre segment de la ligne si elles sont connectées par un chemin passant par des cellules sélectionnées des lignes inférieures. Si la ligne précédente et la ligne courante possèdent une cellule sélectionnée dans la même colonne mais qu'il n'existe pas de lien avec un autre segment de la ligne courante, l'étiquette de liaison sera -1. Une étiquette de liaison égale à -1 signifie donc que le segment auquel appartient la cellule de la ligne courante n'ajoute pas une nouvelle composante connexe. L'étiquette de liaison peut aussi être vide (*None*) s'il n'y a pas de cellule sélectionnée sous cette cellule. La figure 3.4 montre un exemple de noeud ayant des étiquettes de liaison et leur signification.

FIGURE 3.4 – Exemple de noeud et d'étiquettes de liaisons

(a) Noeud dont les deux segments sont liés par les cellules 2 et 4 et où la cellule 6 à une cellule « souterraine ». La cellule 0 n'a pas d'étiquette parce qu'il n'y a pas de cellule sélectionnée sur la même colonne dans la ligne précédente



(b) Représentation visuelle de la signification des étiquettes de liaison



L'algorithme de création de la matrice d'adjacence décrit dans ce chapitre utilise une structure de données pour représenter les noeuds du graphe. Ces derniers étant composés de cellules qui possèdent les 3 attributs suivants :

— Une valeur booléenne indiquant si la cellule est sélectionnée ;

— Le degré* de la cellule si elle est sélectionnée. Si la cellule n'est pas sélectionnée, cette valeur sera 0 ;

— L'étiquette de liaison.

*Cas d'exception : Les cellules isolées (de degré 0) et les cellules de degré 1 n'ayant qu'un élément dessous seront considérées comme des situations identiques. Elles seront donc représentées sous une seule forme, soit celle où le degré est 1. Par exemple, les lignes $[(True, 0, None), (False, 0, None)]$ et $[(True, 1, -1), (False, 0, None)]$ de la figure 3.5 font partie du même noeud représenté par : $[(True, 1, -1), (False, 0, None)]$

FIGURE 3.5 – Exemple d'un cas d'exception où deux lignes différentes (Figures 3.5a et 3.5b) sont considérés comme un seul noeud en utilisant la version où la cellule isolée est de degré 1 (Figure 3.5b)

(a) Ligne $[(True, 0, None), (False, 0, None)]$ (b) Ligne $[(True, 1, -1), (False, 0, None)]$

3.3.2 Génération des noeuds du graphe de génération des forêts de serpents fixes de largeur b

Afin de générer les noeuds composant le graphe de génération des forêts de serpents fixes de largeur b, il est nécessaire de débuter par l'énumération des différentes lignes de base. Une ligne de base est une rangée de cellule contenant uniquement l'information sur les cellules sélectionées (Figure 3.6).

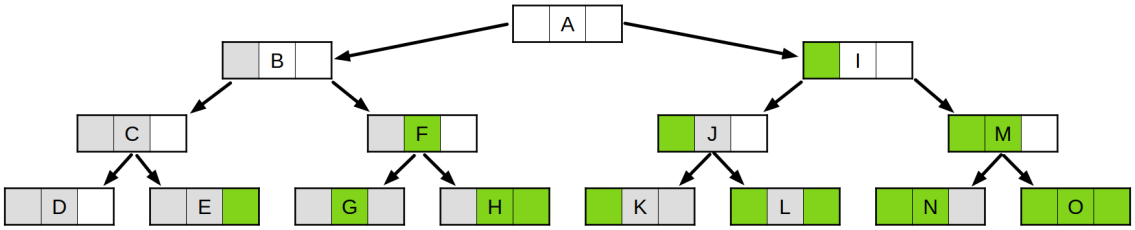
FIGURE 3.6 – Lignes de base d'un rectangle $7 \times h$ où les cellules vertes sont sélectionnées et le chiffre est l'indice de colonne

(a) (b)

La méthode de génération des lignes de base est une méthode récursive débutant avec une ligne contenant uniquement des cellules vides. En débutant avec la cellule la

plus à gauche, la ligne sera dupliquée en ses deux possibilités, soit une avec la cellule sélectionnée et l'autre avec la cellule non sélectionnée. Cette étape est ensuite répétée sur chacune de ces lignes obtenues à partir de la cellule suivante, soit la cellule à droite de la cellule de l'étape précédente. Lorsque toutes les cellules d'une ligne ont été parcourues, celle-ci est ajoutée à la liste des lignes de base. En d'autres mots, la recherche des lignes de base correspond à un arbre binaire enraciné à la ligne vide et où toutes les feuilles sont des lignes de base. Ainsi, il est existe 2^b lignes de base. La figure 3.7 présente l'arbre binaire de la recherche des lignes de base des rectangles de largeur $b = 3$. Les étiquettes alphabétiques des lignes de la figure représente l'ordre de la recherche effectuée.

FIGURE 3.7 – Arbre binaire de génération des lignes de base des rectangles de largeur $b = 3$



Une fois que l'ensemble des lignes de base est complet, il est maintenant possible de déterminer les noeuds du graphe. Pour cela, il faut explorer les différentes possibilités de superposition de ces lignes de base afin d'éviter les cycles et les arbres.

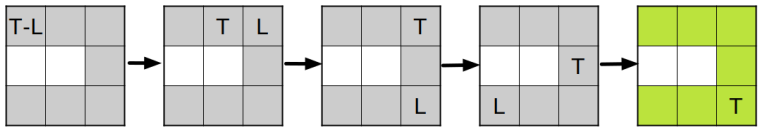
Pour ce faire, toutes les paires de lignes de base superposées sont explorées et les degrés de leurs cellules sont calculés. Pour chacune de ces combinaisons, celle-ci sera rejetée si une de ses cellules est de degré supérieur à 2 ou si un cycle est détecté. Lors de la combinaison des lignes de base, le seul cycle possible est un carré de quatre cellules sélectionnés puisqu'elles auraient toutes un degré égale à 2. Si la combinaison est valide et qu'elle implique des superpositions de cellules sélectionnées, une étiquette de liaison -1 sera ajoutée à chacune de ces cellules sur le noeud.

Afin de détecter les cycles, l'algorithme du lièvre et de la tortue est appliqué à partir

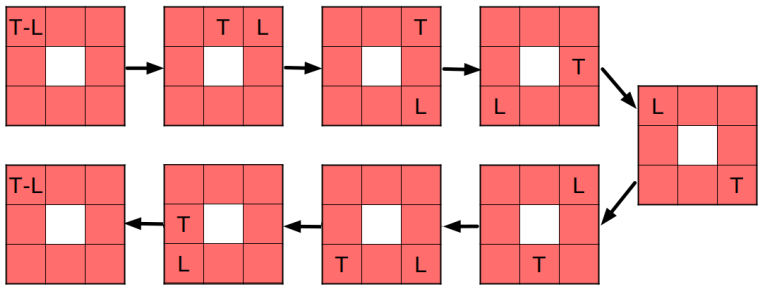
de chacune des cellules de la ligne supérieure. Cet algorithme consiste à parcourir la composante connexe à deux vitesses différentes (Figure 3.8), soit de cellule en cellule (tortue) et en sautant une cellule (lièvre). Ainsi, si la composante connexe contient un cycle, la tortue se fera un jour rattraper par le lièvre puisqu'il aura pris un tour d'avance.

FIGURE 3.8 – Illustration de l'algorithme du lièvre et de la tortue

(a) Le lièvre s'est rendu jusqu'à la fin donc aucun cycle n'a été détecté. Il faut alors recommencer à partir d'une autre cellule de la composante connexe jusqu'à avoir commencé par chacune des cellules de celle-ci.



(b) Le lièvre rattrape la tortue donc il y a un cycle

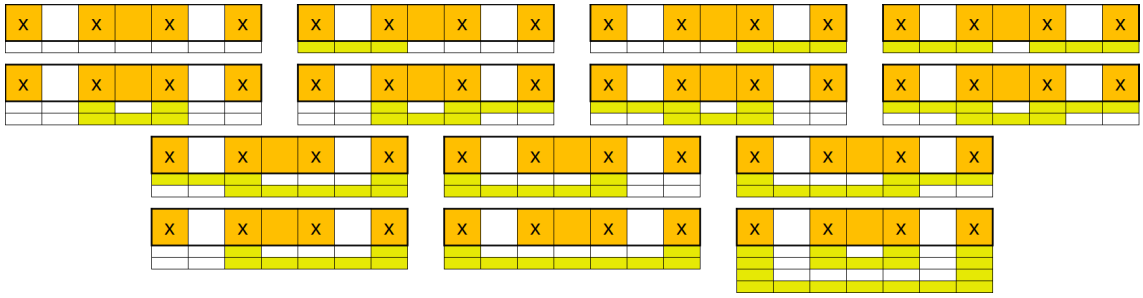


Pour obtenir l'ensemble complet de noeuds, il est aussi important d'établir toutes les possibilités où deux segments non adjacents d'une même ligne font partie de la même composante connexe. Le concept des partitions non croisées permet d'énumérer ces possibilités. Les partitions non croisées, introduites en 1971 par G. Kreweras [20], sont construites à partir d'un ensemble fini et ordonné, ce qui signifie que tous les éléments ont une position fixe les uns par rapport aux autres. Afin d'être non croisées, il est impératif que toutes les liaisons des éléments de l'ensemble ne créent aucune intersection. Le nombre de partitions non croisées possibles pour un ensemble de taille n correspond au n^{e} nombre de Catalan.

Dans notre cas, l'ensemble ordonné est constitué des cellules d'extrémité des segments, soit les cellules de degré inférieur à 2 de la ligne à l'étude. La figure 3.9 présente

un exemple de l'ensemble complet des partitions non croisées de la ligne de base de la figure 3.6b, où les cellules de l'ensemble à partitionner sont marquées par des « x ».

FIGURE 3.9 – Ensemble complet des partitions non croisées de la ligne de base de la figure 3.6b. Notons que ce ne sont pas tous les éléments de cet ensemble qui forment un noeud valide



Étant donné que l'on travaille sur les serpents, les liaisons recherchées font partie des partitions non croisées contenant uniquement des sous-ensembles de taille inférieure ou égale à 2 de l'ensemble à partitionner. Il est impossible de permettre un sous-ensemble contenant plus de deux éléments puisque la structure résultante contiendrait obligatoirement une cellule de degré supérieur à 2, ce qui est interdit. Ainsi, il s'agit d'un sous-ensemble stricte de l'ensemble des partitions non croisées et leur nombre est inférieur au nombre de Catalan. Alors en prenant comme exemple l'une ou l'autre des deux lignes de base de la figure 3.6, l'ensemble des cellules pouvant être lié à d'autres segments est $\{0, 2, 4, 6\}$ et les sous-ensembles sont $\{0, 2\}, \{0, 4\}, \{0, 6\}, \{2, 4\}, \{2, 6\}, \{4, 6\}$. Après avoir éliminé les sous-ensembles contenant les deux extrémités d'un même segment, les sous-ensembles restants peuvent être regroupés selon les règles des partitions non croisées afin de déterminer les différents noeuds possibles.

En pratique, à partir de chaque cellule d'extrémité d'un segment, un algorithme récursif qui couple les cellules itérativement est utilisé pour déterminer les différentes partitions non croisées en ajoutant les étiquettes de liaison correspondantes. Plus précisément, en débutant avec la première cellule sélectionnée à partir de l'indice de colonne 0, la cellule sera couplée itérativement avec toutes les autres cellules sélec-

tionnées de la ligne qui sont de degré inférieur à 2 et qui n'ont pas déjà été couplées. Ces couplages doivent obligatoirement être non croisés puisqu'un croisement nécessite une cellule de degré supérieur à 2, ce qui est impossible dans un serpent. Ils sont effectués de façon récursive jusqu'à ce qu'aucun couplage ne soit possible et que toutes les possibilités aient été explorées. En prenant comme exemple la ligne de base de la figure 3.6b, la figure 3.10 montre les combinaisons valides et la figure 3.11 montre une combinaison invalide puisqu'elle crée un croisement.

FIGURE 3.10 – Exemples de liaisons valides

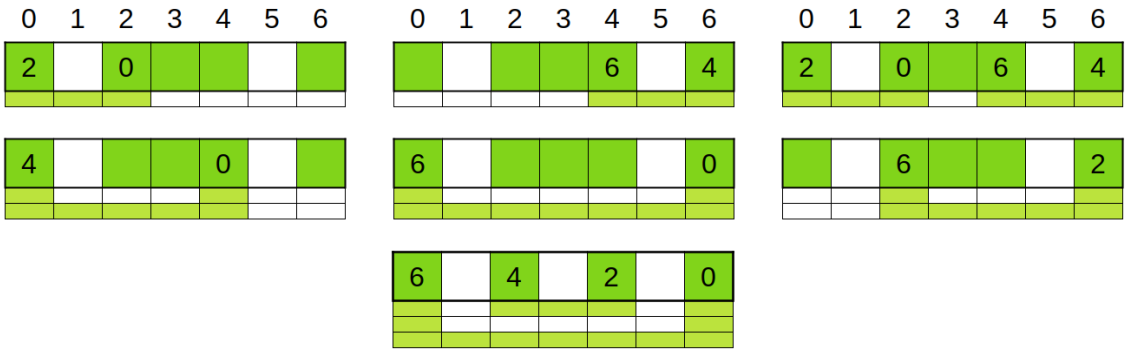
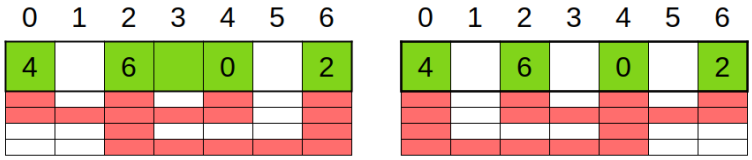
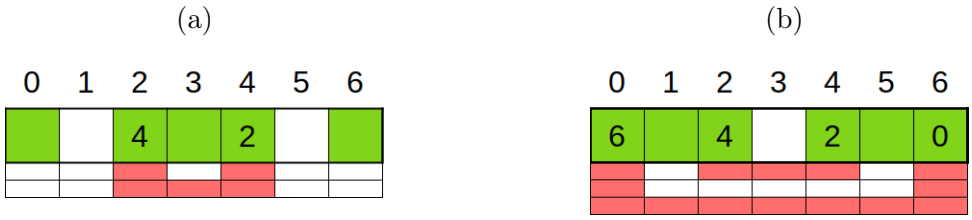


FIGURE 3.11 – Exemples de liaisons invalides à cause d'un croisement



Cet algorithme vérifiera aussi que le noeud résultant ne contient pas de cycle. Cette vérification est nécessaire puisque certains ensembles valides de partitions non croisées peuvent rendre la composante connexe cyclique (Figure 3.12). La figure 3.12b montre un exemple où les règles des partitions non croisées nous permettent d'utiliser la liaison des cellules 0 et 6 en même temps que celle des cellules 2 et 4, ce qui crée un cycle ($\{0, 1, 2, 4, 5, 6, 0, ...\}$). Les cycles sont détectés avec l'algorithme du lièvre et de la tortue où lorsque le bout d'un segment est atteint, un saut est effectué jusqu'à l'indice de colonne noté par l'étiquette de liaison associé au bout du segment.

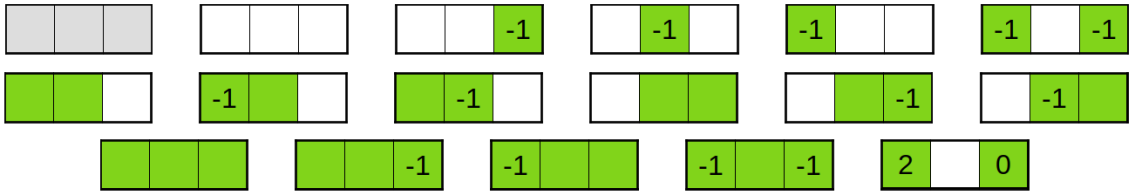
FIGURE 3.12 – Exemples de partitions non croisées valides créant un cycle



En prenant comme exemple la ligne de base de la figure 3.6a, l'algorithme commencera donc en tentant d'effectuer le couplage de la cellule d'indice de colonne 0. La première cellule candidate est la cellule d'indice de colonne 2 et a une étiquette de liaison -1, cependant elle n'est pas éligible au couplage puisqu'un cycle serait créé. La prochaine candidate est la cellule d'indice de colonne 4, qui est éligible, donc une copie du noeud est effectuée en notant les nouvelles étiquettes de liaison des cellules couplées. Par récursivité, le couplage sera continué pour le nouveau noeud en repartant à la cellule d'indice de colonne 0 puisqu'il reste plus de deux cellules candidates au couplage. La cellule d'indice de colonne 2 est disponible pour le couplage mais ne peut être couplée car la seule autre candidate est la cellule d'indice de colonne 6 et cela causerait un croisement. À ce moment, l'algorithme recule d'une étape et reprend son travail à partir du noeud précédent en tentant de coupler la cellule d'indice de colonne 0 à la cellule d'indice de colonne 6 puisque c'est la prochaine candidate de cette étape.

Finalement, en y ajoutant un noeud de départ, l'ensemble résultant de toutes ces opérations est l'ensemble final des noeuds et contient toutes les informations nécessaires à la création de la matrice d'adjacence. La figure 3.13 présente l'ensemble des noeuds appartenant aux rectangles de largeur $b = 3$ où le noeud de départ est représenté par le noeud grisé.

FIGURE 3.13 – Ensemble complet des 17 noeuds du graphe des rectangles de largeur $b = 3$ pour les forêts de serpents



3.3.3 Calcul de la matrice d'adjacence

À partir de l'ensemble de noeuds produit pour les rectangles de largeur b, il est possible de créer un graphe orienté $G = (V, E)$ où V est l'ensemble des noeuds et que $E \subseteq V \times V$ est l'ensemble des arcs. Dans ce graphe représentant la construction des forêts de serpents, le poids d'une arête orientée est un monôme de la forme $y^a z^c$. Ce monôme représente les modifications des caractéristiques de la forêt de serpents en construction. La valeur de chaque caractéristique correspond à son exposant. Ces caractéristiques sont :

— y^a si la ligne ajoutée contient a cellules ;

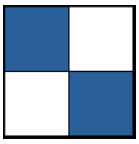
— z^c si c composantes connexes sont ajoutées par la nouvelle ligne. Il est possible que c soit négatif si la ligne vient joindre au moins deux composantes connexes, ce qui réduit le nombre de composantes connexes.

Il est important de noter que d'autres caractéristiques pourraient être ajoutées à ce mônome, bien qu'elles ne soient pas incluses dans ce travail. Par exemple, il serait facile d'ajouter w^k où k représente le nombre de *kisses*, soit le nombre de fois où deux cellules sélectionnées se touchent uniquement par un sommet. La figure 3.14 illustre le concept de *kiss*.

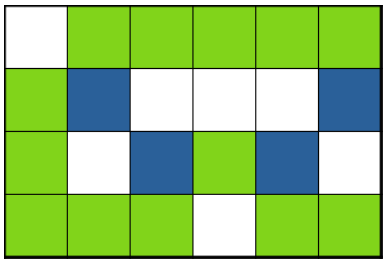
L'algorithme de calcul de l'adjacence entre deux noeuds qui permet d'obtenir ces valeurs est composé de deux fonctions. Le noeud de sortie sera référé ci-dessous par l'appelation ligne/cellule ajoutée. L'algorithme présenté dans cette section est aussi disponible en pseudo-code par l'algorithme 3 à l'annexe A.2. Un exemple de matrice

FIGURE 3.14 – Illustration et exemple du concept de *kiss*

(a) Deux cellules liées entre elles uniquement par un sommet forment un *kiss*



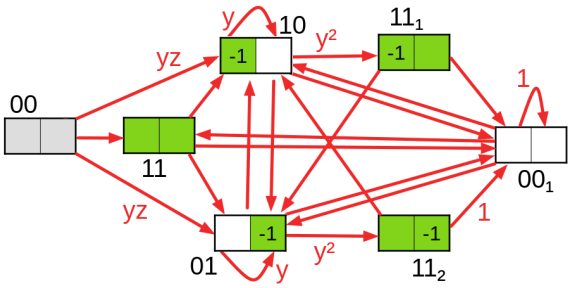
(b) Serpent 6×4 contenant 2 *kisses* (mis en évidence par la couleur bleu)



d'adjacence et du graphe correspondant pour les forêts de serpents pas nécessairement inscrites dans les rectangles de largeur $b = 2$ est présenté à la figure 3.15.

FIGURE 3.15 – Graphe et matrice d'adjacence pour les forêts de serpents des rectangles de largeur $b = 2$

(a) Graphe (voir la matrice d'adjacence pour les poids non illustrés)



(b) Matrice d'adjacence

	00	00_1	01	10	11	11_1	11_2
00	0	0	yz	yz	y^2z	0	0
00_1	0	1	yz	yz	y^2z	0	0
01	0	1	y	yz	0	0	y^2
10	0	1	yz	y	0	y^2	0
11	0	1	y	y	0	0	0
11_1	0	1	y	0	0	0	0
11_2	0	1	0	y	0	0	0

Première fonction (Algorithme 3, lignes 1 à 24)

La première fonction s'assure d'abord que la combinaison des noeuds d'entrée et de sortie ne crée pas de cycle. Ensuite, à partir de l'indice de colonne 0, une vérification est effectuée pour savoir si les deux cellules de la colonne sont compatibles. Si l'une des vérifications de compatibilité échoue, le noeud de sortie n'est pas un successeur du noeud d'entrée et le monôme associé est 0. Cette vérification inclut :

— L'interdiction d'ajouter une cellule sélectionnée sur une cellule de degré 2 ;

— L'interdiction d'ajouter une cellule ayant une étiquette de liaison sur une cellule vide, sauf si la cellule ajoutée est une cellule isolée sur la ligne ;

— L'interdiction d'ajouter une cellule sans étiquette de liaison sur une cellule sélectionnée ;

— La validation de l'étiquette de liaison, si la cellule ajoutée en possède une.

Les figures 3.16 et 3.17 présentent des exemples de vérification réussie et échouée. La valeur de l'étiquette de liaison de chaque cellule du noeud de sortie est indiqué à l'intérieur de celle-ci. Si la cellule ne contient pas de chiffre, c'est qu'elle ne possède pas d'étiquette de liaison.

FIGURE 3.16 – Exemples de vérification réussie

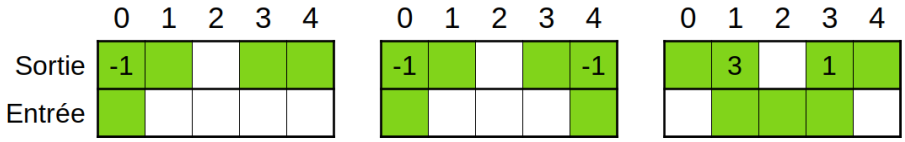
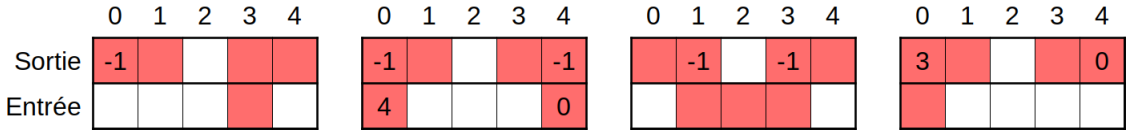
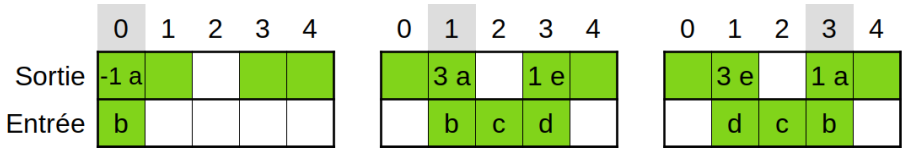


FIGURE 3.17 – Exemples de vérification échouée



La validation de l'étiquette de liaison est effectuée en parcourant la composante connexe jusqu'à sa fin. Le parcours débute par un déplacement vers le bas pour s'assurer d'aller dans la bonne direction. Si l'étiquette de liaison de la cellule ajoutée est -1, celle-ci est valide si l'extrémité de la composante ne se situe pas sur la ligne ajoutée. La figure 3.18 montre plusieurs exemples de ce parcours.

FIGURE 3.18 – Exemples de parcours de validation de l'étiquette de liaison



Cette vérification est effectuée à partir de l'indice de colonne 0 et sera répétée jusqu'à ce que la variable i, qui représente l'indice de colonne présentement vérifié,

soit égale à la longueur du noeud. Lorsque le noeud de sortie contient une cellule sélectionnée à l'indice de colonne où la vérification est effectuée, il s'agit du début d'un nouveau segment et la variable a est incrémentée. Dans ce cas, si la cellule correspondante du noeud d'entrée n'est pas sélectionnée, il est noté qu'il s'agit d'une nouvelle composante. L'algorithme pourra ensuite poursuivre de deux façons en fonction de la variable i.

Si i n'est pas égale à la longueur du noeud, l'algorithme entre dans la portion récursive (deuxième fonction) en incrémentant la valeur de la variable i. Lorsque la portion récursive se termine, la première fonction reçoit les nouvelles valeurs des variables i, a et c et l'algorithme se poursuit en fonction de la valeur de i.

Sinon, la variable c est incrémentée s'il s'agissait d'une nouvelle composante et l'algorithme retourne les variables a et c. Le calcul de l'entrée de la matrice d'adjacence est alors terminé.

Deuxième fonction (Algorithme 3, lignes 25 à 43)

La deuxième fonction est une fonction récursive. Elle commence par effectuer la vérification décrite dans la première fonction. Suite à cette vérification, plusieurs conditions sont vérifiées afin de calculer les valeurs des variables a et c.

S'il est noté que le segment est une nouvelle composante connexe et que la cellule ajoutée n'est pas sélectionnée, la variable c est incrémentée. S'il est noté que le segment est une nouvelle composante connexe et que les deux cellules de la colonne d'extrémité droite du segment sont sélectionnées, il ne s'agit donc pas d'une nouvelle composante et la variable c ne sera pas modifiée.

S'il est noté que le segment n'est pas une nouvelle composante et que les deux

cellules de la colonne sont sélectionnées, il s'agit donc de l'union de deux composantes et la variable c est décrémentée.

Ensuite, si la cellule ajoutée n'est pas sélectionnée, le segment est terminé et la deuxième fonction retourne les variables a, c ainsi que la variable i incrémentée de 1. L'algorithme reprend alors à la première fonction à partir de l'indice de colonne i reçu de la deuxième fonction.

Si la cellule ajoutée est sélectionnée et qu'il existe une colonne suivante, l'algorithme répète la deuxième fonction à partir de la colonne suivante et retourne son résultat à la première fonction. Par contre, s'il n'existe pas de colonne suivante, la valeur c est incrémentée s'il s'agissait d'une nouvelle composante et que la cellule d'entrée n'est pas sélectionnée et l'algorithme retourne les variables a, c ainsi que la variable i incrémentée de 1.

3.3.4 Utilisation de la matrice d'adjacence

Une fois que la matrice d'adjacence est complétée, il est possible de l'utiliser pour obtenir deux types d'informations :

— L'ensemble des forêts de serpents du rectangle de largeur b ;

— La série génératrice des forêts de serpents de largeur b.

Production de la série génératrice

La série génératrice pouvant être calculée à partir de la matrice d'adjacence permet d'obtenir le nombre d'exemplaires ayant les mêmes caractéristiques (voir la sous-section 3.3.3). Pour ce faire, la méthode des matrices de transfert [15][16][17][18] doit être utilisée.

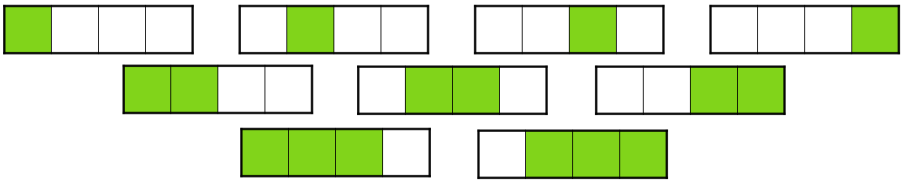
Prenons en exemple la matrice d'adjacence pour les forêts de serpents des rectangles de largeur $b = 2$ présentée à la figure 3.15b et nommée $A_{2\times h}$ dans l'équation 3.1.

$$A_{2\times h} = \begin{bmatrix} 0 & 0 & yz & yz & y^2z & 0 & 0 \\ 0 & 1 & yz & yz & y^2z & 0 & 0 \\ 0 & 1 & y & yz & 0 & 0 & y^2 \\ 0 & 1 & yz & y & 0 & y^2 & 0 \\ 0 & 1 & y & y & 0 & 0 & 0 \\ 0 & 1 & y & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & y & 0 & 0 & 0 \end{bmatrix} \tag{3.1}$$

Il est possible d'obtenir la série génératrice en débutant par les opérations matricielles $(I_7 - x \times A_{2\times h})^{-1}$, où I_7 est la matrice identité de format 7×7 et l'exposant de x sera la hauteur h du rectangle dans les monômes du développement de la série génératrice. Afin d'inclure toutes les possibilités qui ont au moins une cellule sur la ligne du bas et la ligne du haut du rectangle, les éléments $[1, j]$, où $3 \leq j \leq 7$, de la matrice $(I_7 - x \times A_{2\times h})^{-1}$ doivent ensuite être développés en série de Taylor et leurs séries doivent être additionnées. La série résultante peut être développée afin d'obtenir la série génératrice des forêts de serpents du rectangle. L'ensemble des termes de cette série où l'exposant de z est 1 correspond à la série génératrice des serpents du rectangle.

Toutefois, il est possible d'améliorer cette série génératrice pour qu'elle devienne la série génératrice des serpents et des forêts inscrits dans leurs rectangle en soustrayant à plusieurs reprises les séries génératrice des rectangles de bases (b) inférieures. Le nombre de soustractions correspond au nombre de façons d'insérer le plus petit rectangle dans le plus grand (voir la figure 3.19).

FIGURE 3.19 – Un rectangle de largeur $b = 4$ contient 4 copies du rectangle de largeur $b = 1$, 3 copies du rectangle de largeur $b = 2$ et 2 copies du rectangle de largeur $b = 3$



Pour l'exemple actuel, les rectangles $b = 2$ peuvent contenir les rectangles $b = 1$ de deux façons différentes donc il faut soustraire deux fois la série génératrice des serpents et des forêts des rectangles $b = 1$ afin d'obtenir la série génératrice des serpents et des forêts inscrits présentée par l'équation 3.2. La série génératrice des forêts de serpents inscrites dans un rectangle de largeur $b = 1$ est obtenue à partir de la matrice d'adjacence présentée dans la figure 3.20. Les lignes contenues entre les équations 3.3 à 3.4 montrent différentes étapes du calcul de cette série.

$$\begin{aligned} serie_{2\times h} &= \sum_{j=3}^{7} (I_7 - x \times A_{2\times h})^{-1}[1,j] - 2 \times serie_{1\times h} \\ &= xy^2z + 2x^2y^2z^2 + 4x^2y^3z + 2x^3y^2z^2 + 8x^3y^3z^2 + 2x^3y^3z^3 + 6x^3y^4z \\ &\quad + 5x^3y^4z^2 + 2x^3y^5z + O\left(x^4\right) \end{aligned} \tag{3.2}$$

$$serie_{1xh} = \left(I_3 - x \times \begin{bmatrix} 0 & 0 & yz \\ 0 & 1 & yz \\ 0 & 1 & y \end{bmatrix} \right)^{-1} [1,3] \tag{3.3}$$

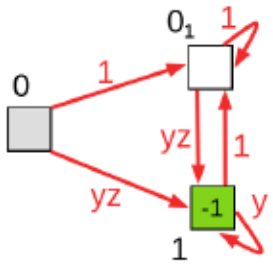
$$= \begin{bmatrix} 1 & \frac{x^2yz}{-x^2yz+x^2y-xy-x+1} & \frac{-x^2yz+xyz}{-x^2yz+x^2y-xy-x+1} \\ 0 & \frac{-xy+1}{-x^2yz+x^2y-xy-x+1} & \frac{xyz}{-x^2yz+x^2y-xy-x+1} \\ 0 & \frac{x}{-x^2yz+x^2y-xy-x+1} & \frac{1-x}{-x^2yz+x^2y-xy-x+1} \end{bmatrix} [1,3]$$

$$= \frac{-x^2yz + xyz}{-x^2yz + x^2y - xy - x + 1}$$

$$= xyz + x^2y^2z + x^3y^2z^2 + x^3y^3z + O\left(x^4\right) \tag{3.4}$$

FIGURE 3.20 – Graphe et matrice d'adjacence pour les forêts de serpents des rectangles de largeur $b = 1$

(a) Graphe



(b) Matrice d'adjacence

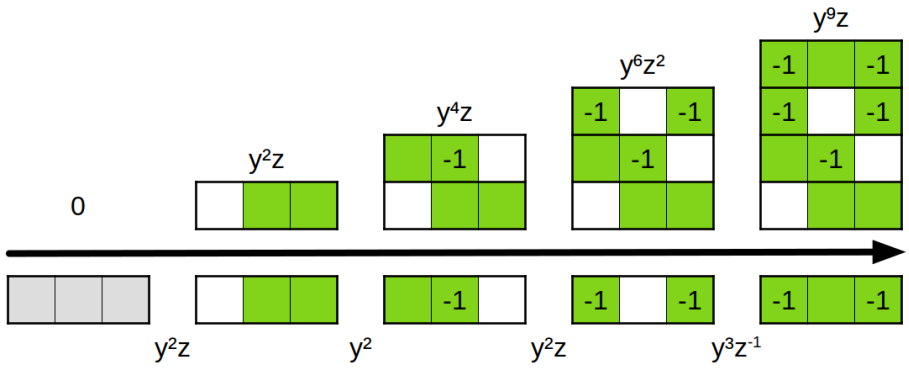
	0	0_1	1
0	0	0	yz
0_1	0	1	yz
1	0	1	y

Énumération des serpents maximaux et des forêts de serpents maximales

Pour obtenir l'ensemble des serpents ou des forêts de serpents de taille maximale dans un rectangle R donné, on effectue une fouille en profondeur du graphe correspondant. Cette recherche doit être effectuée à partir du noeud de départ et parcourir un nombre de noeuds égal à la hauteur h du rectangle R. L'évolution des caractéristiques décrites à la section 3.3.3 peut être suivie en multipliant les monômes à chaque déplacement le long d'un arc dans le graphe. La figure 3.21 présente un exemple de construction d'un rectangle en se déplaçant dans le graphe. Bien que dans cette figure

le rectangle construit soit un serpent maximal, le résultat d'un parcours de longueur h du graphe peut produire n'importe quelle forêt de serpents contenue dans un rectangle de dimensions $b \times h$.

FIGURE 3.21 – Construction d'un rectangle en se déplaçant dans le graphe



En procédant de cette façon, lors de la recherche des serpents maximaux, il n'y a qu'à s'assurer que le monôme final possède une valeur de c, l'exposant de z, égale à 1. Afin d'obtenir l'ensemble de serpents (ou de forêts de serpents) d'aire maximale, l'objectif est de ne conserver que ceux qui ont la plus grande valeur de a, l'exposant de y. De plus, si A est la matrice d'adjacence, alors l'entrée en position $[i, j]$ de la matrice A^h donne le nombre de forêts de serpents de hauteur h qui commencent par le noeud i et se terminent par le noeud j.

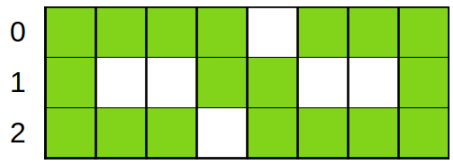
L'avantage principal de cette méthode comparativement à celle de la section 3.2 est qu'il est possible d'y ajouter de nombreux critères d'élagage permettant d'accélérer la recherche. Afin de favoriser leur utilisation, il est préférable de trier les noeuds par ordre décroissant de a afin que la recherche soit d'abord effectuée sur les noeuds ayant un plus gros impact sur la taille du résultat, puisque cela permet généralement de trouver un serpent ou une forêt de taille maximale plus tôt dans la recherche. Toutefois, la majorité de ces critères n'ont pas encore été prouvés et ne peuvent donc pas être utilisés de façon certaine. En voici quelques exemples :

— S'il était démontré que lorsque la taille d'une forêt maximale de la dimension de la zone de recherche restante additionnée à la taille de la forêt en cours de

construction est inférieure à la taille maximale trouvée dans un autre chemin, il ne serait plus nécessaire de poursuivre ce chemin. La figure 3.22 présente un exemple de cette situation. Ainsi, il serait possible d'utiliser un tableau des tailles précédemment calculées pour considérablement accélérer la recherche.

FIGURE 3.22 – Exemple où il ne serait plus nécessaire de poursuivre la recherche sur un chemin

(a) Serpent de longueur 18 précédement obtenu dans la recherche



(b) Exemple d'une profondeur de 4 noeuds où la taille de la forêt maximale (9) pouvant entrer dans l'espace n'est pas suffisante pour dépasser le résultat précédent

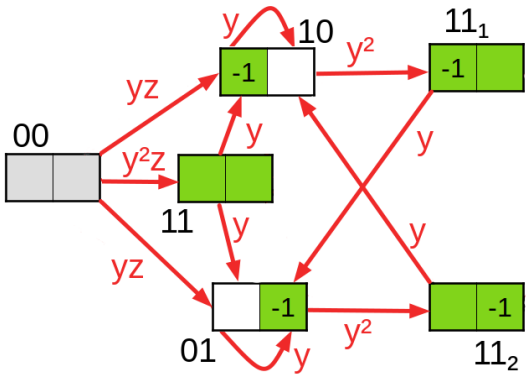
— Si une borne inférieure concernant le nombre minimal de cellules sélectionnées sur une ligne était établie et démontrée, il serait possible de ne plus rechercher tous les chemins passant par un noeud du graphe ayant une valeur de a inférieure à cette borne. Par exemple, il a été démontré que pour la largeur $b = 3$, le nombre de cellules sélectionnées de chaque ligne doit être supérieur à 2. Une borne plus générale pourrait aussi être obtenue en démontrant la conjecture selon laquelle, dans les forêts de serpents maximales d'un rectangle de largeur b, toutes les lignes du rectangle possèdent au moins $\lfloor(2/3)b\rfloor - 3$ cellules sélectionnées [21].

— S'il était démontré qu'il n'existe aucun serpent ou forêt maximal contenant une ligne à n cellules, il serait possible de ne pas rechercher tous les chemins passant par un noeud du graphe ayant une valeur de $a = n$. Par exemple, il a été démontré que dans une forêt maximale contenue dans un rectangle $3 \times h$, il n'existe pas de ligne avec exactement une cellule sélectionnée mais il existe des lignes à zéro cellules sélectionnées [21]. Donc pour trouver les forêts maximales des rectangles $3 \times h$, il n'est pas nécessaire d'explorer les chemins passant par un noeud où $a = 1$.

Finalement, il serait aussi possible d'améliorer la recherche des serpents maximaux en simplifiant la matrice d'adjacence calculée précédemment pour éliminer les noeuds et les liens d'adjacence qui ne permettent pas de terminer avec une seule composante connexe. Cette simplification n'a pas été explorée lors de ce travail mais pourrait faire l'objet de travaux futurs.

FIGURE 3.23 – Graphe et matrice d'adjacence simplifiés pour les serpents à translation près des rectangles de largeur $b = 2$

(a) Graphe



(b) Matrice d'adjacence

	00	01	10	11	11_1	11_2
00	0	yz	yz	y^2z	0	0
01	0	y	0	0	0	y^2
10	0	0	y	0	y^2	0
11	0	y	y	0	0	0
11_1	0	y	0	0	0	0
11_2	0	0	y	0	0	0

Aires des serpents et des forêts maximales

Suite à l'implémentation de la méthode des matrices de transfert présentée dans cette section, l'aire maximale des serpents et des forêts de serpents de plusieurs rectangles a été calculée. La suite obtenue pour l'aire des serpents maximaux dans les carrés correspond à celle définit dans OEIS [22]. Les tableaux 3.1 et 3.2 contiennent ces résultats. De plus, les tableaux présentant le nombre de serpents maximaux et de forêts maximales libres et fixes dans ces rectangles sont inclus dans l'annexe B.

TABLE 3.1 – Aires des serpents maximaux par dimensions de rectangle

h \ b	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	1							
2	2	3						
3	3	5	7					
4	4	6	9	11				
5	5	8	11	14	17			
6	6	9	14	17	21	24		
7	7	11	16	20	24	29	33	
8	8	12	18	22	27	32	38	42
9	9	14	20	25	30	36	42	48
10	10	15	22	28	34	40	46	
11	11	17	24	30	37	44	50	
12	12	18	26	33	40	47	55	
13	13	20	28	36	43	51	59	
14	14	21	30	38	46	55		
15	15	23	32	41	49	59		
16	16	24	34	44	52	62		
17	17	26	36	46	55	66		
18	18	27	38	49	58	70		
19	19	29	40	52	61	74		
20	20	30	42	54	65			
21	21	32	44	57				
22	22	33	46	60				
23	23	35	48	62				
24	24	36	50	65				
25	25	38	52	68				
26	26	39	54	70				
27	27	41	56	73				
28	28	42	58	76				
29	29	44	60	78				
30	30	45	62	81				
31	31	47	64	84				
32	32	48	66	86				
33	33	50	68	89				
34	34	51	70	92				
35	35	53	72	94				
36	36	54	74	97				
37	37	56	76	100				
38	38	57	78	102				
39	39	59	80	105				
40	40	60	82	108				

TABLE 3.2 – Aires des forêts de serpents maximales par dimensions de rectangles

h \ b	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	1									
2	2	3								
3	3	5	7							
4	4	6	9	11						
5	5	8	11	14	17					
6	6	9	14	17	21	25				
7	7	11	16	20	24	29	33			
8	8	12	18	22	28	33	38	43		
9	9	14	20	25	31	38	43	49	55	
10	10	15	22	28	34	41	48	54	61	67
11	11	17	24	30	38	45	52	60	67	74
12	12	18	26	33	41	50	57	65	74	81
13	13	20	28	36	44	53	62	70	79	88
14	14	21	30	38	48	57	66	76	85	94
15	15	23	32	41	51	62	71	81	92	101
16	16	24	34	44	54	65	76	86	97	108
17	17	26	36	46	58	69	80	92	103	114
18	18	27	38	49	61	74	85	97	110	121
19	19	29	40	52	64	77	90	102	115	
20	20	30	42	54	68	81	94	108	121	
21	21	32	44	57	71	86	99	113	128	
22	22	33	46	60	74	89	104	118	133	
23	23	35	48	62	78	93	108	124	139	
24	24	36	50	65	81	98	113	129	146	
25	25	38	52	68	84	101	118	134	151	
26	26	39	54	70	88	105	122	140	157	
27	27	41	56	73	91	110	127	145	164	
28	28	42	58	76	94	113	132	150	169	
29	29	44	60	78	98	117	136	156	175	
30	30	45	62	81	101	122	141	161	182	
31	31	47	64	84	104	125	146	166	187	
32	32	48	66	86	108	129	150	172	193	
33	33	50	68	89	111	134	155	177	200	
34	34	51	70	92	114	137	160	182	205	
35	35	53	72	94	118	141	164	188	211	
36	36	54	74	97	121	146	169	193	218	
37	37	56	76	100	124	149	174	198	223	
38	38	57	78	102	128	153	178	204	229	
39	39	59	80	105	131	158	183	209	236	
40	40	60	82	108	134	161	188	214	241	

Chapitre 4

Application des réseaux neuronaux

L'objectif de l'application des réseaux neuronaux dans le cadre de ce mémoire est la génération de serpents maximaux dans des rectangles de dimensions supérieures à ceux faisant partie de l'ensemble d'entraînement. Cette expérimentation n'a été tentée que pour la classe des serpents maximaux et non pour les forêts maximales.

4.1 Motivations

La première motivation vient de l'hypothèse que, lors de leur construction dans les rectangles, les serpents maximaux satisfont certaines contraintes géométriques et possèdent une structure qui n'a pas encore été établie. En effet, des langages formels et des séries génératrices à plusieurs variables ont été établis pour la construction des forêts de serpents maximales des rectangles $3 \times h$ lorsque $h \geq 6$, $4 \times (4 + 3k)$ et $5 \times (5 + 3k)$ où k est un nombre entier supérieur à 0 [21]. Les réseaux neuronaux sont très efficaces dans les tâches de reconnaissance de patrons [23][24]. Ainsi, il est peut-être possible de construire un modèle de réseaux neuronaux capable de découvrir

et d'apprendre ces patrons afin de les utiliser pour concevoir les serpents maximaux de rectangles trop grands pour être calculables à l'aide des algorithmes exhaustifs d'énumération.

La deuxième motivation vient de la complexité des algorithmes de recherche des serpents et des forêts maximales. Effectivement, la complexité de calcul des algorithmes énumératifs augmente de façon exponentielle en fonction de la taille du rectangle. Cela rend ces méthodes limitées et inutilisables avec les grands rectangles. S'il était possible d'entraîner un réseau neuronal à reconnaître les patrons géométriques des serpents maximaux et des forêts de serpents maximales et à les reproduire dans des rectangles plus grand que l'ensemble d'entraînement, ces résultats pourraient être utilisés dans l'étude de ces structures mathématiques.

La dernière motivation est qu'il s'agit, à notre connaissance, de la première fois que des réseaux neuronaux sont utilisés dans une investigation combinatoire d'une famille de polyominos ou de chemins auto-évitants.

4.2 Revue de littérature

Bien qu'il s'agisse d'une première utilisation des réseaux neuronaux dans une étude sur les chemins auto-évitants, la littérature sur les différents types de réseaux neuronaux et leurs multiples utilisations est abondante. De plus, la littérature sur les chemins auto-évitants comporte beaucoup d'applications dans le domaine du pliage des protéines [25][26][27] et de molécules, un domaine où les réseaux neuronaux sont de plus en plus utilisés [28].

Il existe de nombreuses architectures de réseaux neuronaux présentant tous leurs avantages et leurs inconvénients. Afin de favoriser les chances de réussite, il est im-

portant de choisir une architecture qui corresponde le mieux à l'objectif défini au début de ce chapitre, soit la génération de serpents maximaux dans des rectangles de dimensions supérieures à ceux faisant partie de l'ensemble d'entraînement. À notre connaissance, tous les algorithmes d'apprentissage machine utilisent des environnements fixes pour l'apprentissage et les tests. Par exemple, AlphaGo Zero [29] utilise un environnement fixe correspondant à la taille de la planche de jeu de Go. Dans le cas de notre projet, les dimensions des rectangles sont variables, ce qui pose un défi supplémentaire majeur.

Les types de réseaux de neurones et d'apprentissage machine ayant été considérés pour cette tâche sont les réseaux récurrents à mémoire à court et long terme, les auto-encodeurs, les *Pointer Networks*, l'apprentissage par renforcement ainsi que les réseaux antagonistes génératifs.

La premier type de réseau envisagé a été celui des réseaux récurrents à mémoire court et long terme, ou *Long-Short Term Memory* (LSTM) en anglais. Dans un réseau de neurones récurrents, les entrées récentes ont un impact sur le traitement des entrées suivantes sans ajuster les poids du réseau. L'architecture LSTM a été introduite en 1997 par Hochreiter et Schmidhuber [30] afin d'améliorer la performance des réseaux de neurones récurrents pour les problèmes contenant des données séquentielles nécessitant des informations éloignées dans la séquence. La principale contrainte des réseaux LSTM est qu'ils nécessitent des tailles fixes de données d'entrée et de sortie. Ce type de réseau neuronal est déjà particulièrement exigeant en calcul [31] et l'utilisation d'une quantité considérable de bourrage des rectangles d'entraînement serait nécessaire afin que le réseau puisse prédire des résultats pour les rectangles de plus grandes dimensions. Ce bourrage augmenterait le nombre de cellules des rectangles d'entraînement à l'aide de valeurs neutres telles que 0 si 1 signifie « sélectionnée » et -1 signifie « non sélectionnée », et ce jusqu'à ce que la taille des rectangles atteigne la taille d'entrée du réseaux. Cela pourrait nuire à la performance du réseau en créant un déséquilibre dans l'ajustement des poids ou en allongeant de façon considérable

l'entraînement pour tenter de réduire ce déséquilibre.

En se rapportant au domaine des molécules, la littérature contient de nombreuses publications utilisant différents auto-encodeurs [32][33], un type de réseau neuronal qui apprend de manière non supervisée. Pour créer une représentation de l'ensemble des données, l'auto-encodeur compresse et décompresse les données. Il est cependant difficile de ramener ces méthodes au problème actuel puisqu'elles utilisent des données d'entrée riches en attribut pouvant servir à l'apprentissage. Malheureusement, en utilisant les cellules du rectangle de départ comme données d'entrée, il est facile de se rendre compte du manque d'attribut qu'elles possèdent puisque toutes les cellules de départ sont vides et identiques (à l'exception des cellules adjacentes au périmètre du rectangle).

Un autre type d'auto-encodeur variable intéressant est celui de Zhang et al. [34] qui est spécifique aux graphes orientés et acycliques. Il ne peut cependant pas être utilisé dans le cadre de ce travail puisque bien que les serpents maximaux soient acycliques, les graphes (voir section 3.3) permettant de les générer peuvent contenir des cycles.

Les *Pointer Networks* [35] sont un autre type de réseau permettant de résoudre des problèmes de parcours de graphe. Ils ont donc été envisagés pour prédire les serpents maximaux et les forêts maximales à partir des graphes d'adjacence, ou automates, décrits dans la section 3.3. Ces réseaux ont été appliqués avec succès à d'autres problèmes de graphes NP-Difficiles comme le problème du voyageur de commerce, où l'on souhaite trouver le plus petit cycle hamiltonien dans un graphe dont les arêtes sont pondérées. L'avantage principal de ce type de réseau est qu'il n'utilise pas des dimensions fixes de données d'entrées et de sorties, ce qui permet d'entraîner le modèle sur des petits graphes afin de prédire les résultats sur des graphes plus grands [36]. Malheureusement, le problème actuel impose des contraintes additionnelles et, selon la revue de littérature effectuée, les *Pointer Networks* ne semblent pas avoir été utilisés dans des problèmes contenant une de ces contraintes :

— Un contrôle précis de la longueur de séquence ciblée correspondant à la hauteur h du rectangle afin d'obtenir une réponse optimale pour cette longueur. Le réseau doit être capable de prévoir l'arrêt de la séquence afin de ne pas s'arrêter sur un noeud intermédiaire d'un séquence prévue pour être plus longue.

— L'existence de plusieurs bonnes réponses.

Il est aussi possible de voir le problème du serpents dans la boîte comme un jeu dont le but est de créer le plus long serpent. L'apprentissage par renforcement [37], ou *Reinforcement Learning* (RL) en anglais, a donc été considéré puisque cette méthode offre de très bonnes performances dans le domaine des jeux [38][39]. Cette méthode fonctionne à l'aide de récompenses déterminées en fonction de la qualité des résultats obtenus et le modèle apprend en tentant de maximiser cette récompense. Un des principaux inconvénients de l'apprentissage par renforcement est qu'il nécessite un système fermé, ce qui veut dire que les états, les actions et la taille de l'environnement observable sont établis à la création du modèle et demeurent inchangés. Un autre exemple de cette méthode est le *Graph Convolutional Policy Network* (GCPN) [40], un modèle permettant la génération de nouvelles molécules à l'aide d'apprentissage par renforcement. Il nécessite un environnement de génération fixe où chaque élément possède de nombreuses propriétés favorisant la conception de la récompense. Dans le cas du problème actuel où le but est d'explorer des rectangles de plus en plus grands, il est difficile de croire qu'un environnement fixe permettrait d'entraîner un modèle capable de résoudre différentes tailles de rectangles. Bien qu'il serait peut-être possible de concevoir un modèle possédant un champ de vision fixe, le manque de propriétés significatives des cellules composant les serpents maximaux est donc un autre défi important pour cette méthode.

Un autre type de réseau très populaire, particulièrement dans la génération d'images, est le réseau antagoniste génératif [41][42], ou *Generative Adversarial Network* (GAN) en anglais. Ce modèle est basé sur la compétition entre deux réseaux neuronaux, soit un générateur et un discriminateur. Lors de l'entraînement, le discriminateur reçoit des échantillons de données provenant du générateur et de l'ensemble d'entraînement.

Son objectif est de déterminer correctement pour chaque échantillon reçu s'il provient du générateur ou de l'ensemble d'entraînement. Quant au générateur, son objectif est de produire des échantillons capables de tromper le discriminateur. Lorsque le taux de succès du discriminateur converge vers 50%, équivalent à deviner aléatoirement, le discriminateur n'est plus capable de distinguer la provenance des échantillons. Dans ce cas, les échantillons produits par le générateur devraient ressembler de manière significative aux données d'entraînement, bien que ce ne soit pas toujours le cas [43]. Cela est une des raisons pour lesquelles les GANs sont difficiles à entraîner et que l'entraînement est parfois instable. Cependant, il existe diverses méthodes permettant d'améliorer la stabilité de l'entraînement des GANs telles que l'algorithme de Wasserstein [44] qui modifie principalement la fonction de perte du GAN afin de réduire la distance entre la distribution des images générées et celles des images d'entraînement. Ce type de réseau possède lui aussi la contrainte de devoir utiliser des tailles fixes de données d'entrée et de sortie. Par contre, leurs excellents résultats dans le domaine de la génération d'images ont inspiré l'idée de méthodologie de ce chapitre permettant de contourner cette limitation. De plus, il existe de nombreuses versions de GANs [45] possédant différentes forces et faiblesses, ce qui offre un large éventail d'idées d'ajustements possibles pour le projet actuel. Le réseau antagoniste génératif conditionnel (*Conditional GAN*) [46] et le réseau antagoniste contraint (*Constrained Adversarial Network*) [47], par exemple, sont des réseaux permettant d'influencer la génération à l'aide de classification ou de contraintes structurelles. Dans ce cas-ci, cela pourrait s'avérer utile afin de s'assurer que les rectangles produits sont de la bonne dimension.

4.3 Generative Adversarial Networks (GAN)

4.3.1 Justification du choix

Le choix d'utiliser un GAN dans le cadre de ce travail a été motivé par plusieurs avantages. Ces avantages proviennent parfois directement du type de réseau, mais certains avantages découlent aussi de la méthodologie qu'il permet.

Évidemment, un des avantages les plus attractifs des GANs est leur capacité de génération. Dans le cadre du projet actuel, cette capacité est indispensable puisque l'objectif est de permettre la génération de serpents maximaux dans des rectangles plus grands que les rectangles d'entraînement. Les performances des GANs ont été démontrées avec succès dans plusieurs problèmes de génération d'images. C'est donc cette propriété qui a été la motivation principale de la méthodologie choisie dans ce chapitre. Cette idée, discutée en détail dans la section suivante, consiste à convertir les serpents maximaux de différentes dimensions de rectangles en images ayant toutes les mêmes dimensions afin de tenter de généraliser le réseau.

Un autre avantage de la méthode utilisant un GAN est qu'elle ne dépend pas des graphes d'adjacence présentés dans la section 3.3, dont la création et l'utilisation nécessite une grande quantité de mémoire et un temps computationnel considérable, augmentant de façon exponentielle en fonction de la largeur b. Ainsi, dans le cas d'un succès de la méthodologie actuelle, il serait peut-être possible de produire des tailles de rectangles de dimension b plus élevée et plus efficacement. Toutefois, cela nécessiterait que les patrons de ces serpents soient semblables à ceux des largeurs inférieures utilisées pour l'entraînement.

4.3.2 Conversion des serpents en images

Afin de tenter de produire un modèle général permettant l'entraînement et la prédiction d'une grande variété de dimensions de rectangles, les serpents maximaux des rectangles $b \times h$ sont convertis en images de dimension fixe et supérieure aux dimensions maximales des rectangles à prédire. Ainsi, si la taille des images données en entrée et en sortie est $m \times n$, cette image peut représenter un rectangle de dimensions $b \times h$ où $1 \leq b \leq m$ et $1 \leq h \leq n$. Le terme **rectangle** est utilisé lorsque les unités de longueur des dimensions sont données en cellules alors que le terme **image** est utilisé pour des dimensions données en pixels.

Dans le cadre de ce chapitre, le blanc signifie que le pixel ou la cellule est sélectionné et le noir signifie non sélectionné.

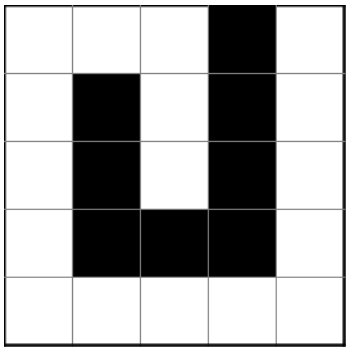
Conversion des rectangles en images

Pour effectuer la représentation d'un rectangle de dimensions $b \times h$ cellules dans une image de dimensions $m \times n$ pixels, il suffit de représenter chaque cellule par plusieurs pixels. Cette conversion sera effectuée en effectuant la division longue de chaque dimension de l'image (donnée en pixels) par la dimension correspondante du rectangle (donnée en cellules).

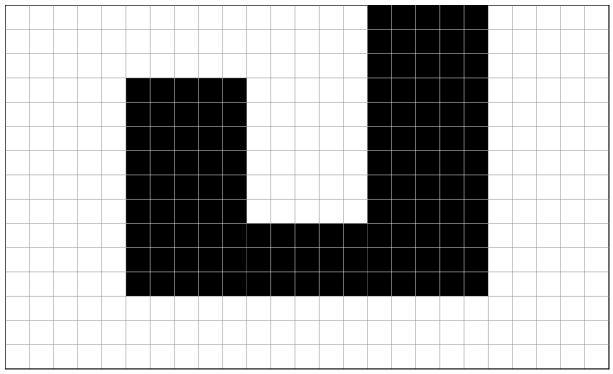
Si les dimensions de rectangle sont des diviseurs des dimensions de l'image, cela revient simplement à représenter chaque cellule par un nombre de pixels égal au quotient de cette division pour chaque dimension. Par exemple, pour convertir un rectangle 5×5 en image 25×15, chaque cellule sera représentée par un ensemble de pixels de dimensions 5×3. La figure 4.1 illustre l'exemple précédent.

FIGURE 4.1 – Exemple de conversion où les dimensions sont divisibles

(a) Serpent maximal d'un rectangle 5×5



(b) Image de dimensions 25×15 représentant un serpent maximal du rectangle 5×5



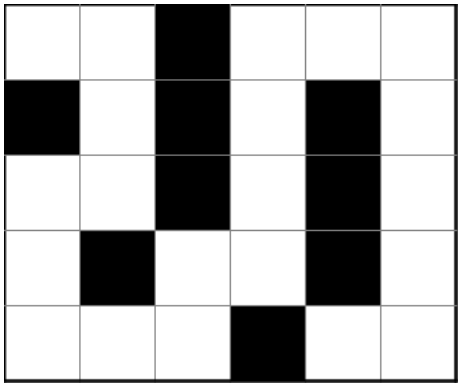
Dans le cas où une dimension du rectangle n'est pas un diviseur entier de la dimension correspondante de l'image, le reste de la division donne le nombre de pixels qui seront répartis, un pixel par cellule, pour obtenir le nombre total de pixels désiré. Cette répartition est effectuée en partant de la gauche s'il s'agit de la largeur ou du haut s'il s'agit de la hauteur. Ainsi, pour transformer un rectangle de dimensions 6×5 en une image de dimensions 25×18, la première colonne du rectangle aura une largeur de 5 pixels alors que les autres colonnes auront une largeur de 4 pixels et les trois premières rangées auront une hauteur de 4 pixels alors que les autres rangées auront une hauteur de 3 pixels. Il y aura donc quatre formats de cellule dans l'image résultante, soit 3 cellules de 5×4 pixels, 2 cellules de 5×3 pixels, 15 cellules de 4×4 pixels et 10 cellules de 4×3 pixels. La figure 4.2 illustre cet exemple.

Conversion des images en rectangles

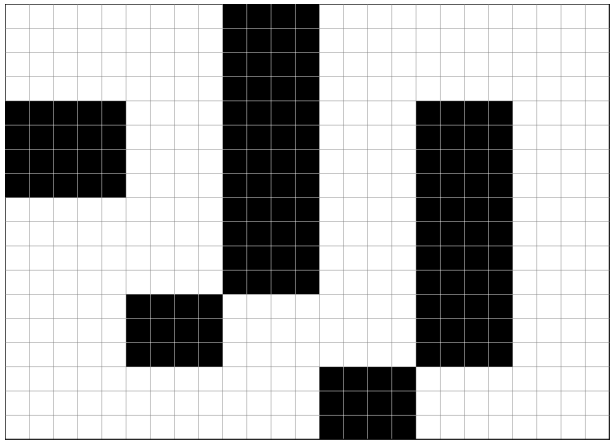
Évidemment, la conversion cellule-pixels décrite ci-haut n'a d'utilité que s'il est possible d'effectuer la conversion en sens inverse et de retrouver le rectangle de taille original. Pour cela, il faut déterminer les dimensions du rectangle désiré. Ensuite, la valeur moyenne de l'ensemble des valeurs numériques des pixels représentant une cellule est utilisée afin de déterminer si cette cellule est sélectionnée ou non dans

FIGURE 4.2 – Exemple de conversion où le quotient des dimensions possède un reste non nul

(a) Serpent maximal d'un rectangle 6×5



(b) Image de dimensions 25×18 représentant un serpent maximal du rectangle 6×5



le rectangle final. Ces valeurs numériques peuvent être $[0, 1]$ dans le cas d'images contenant strictement des pixels noirs et blancs ou dans l'intervalle $[0, 255]$ dans le cas d'images pouvant contenir des pixels gris.

Cette méthode se prête parfaitement au problème actuel puisqu'elle permet de retrouver le serpent représenté dans l'image même si celui-ci contient des pixels ayant une valeur erronée. Effectivement, si un pixel a une valeur signifiant que la cellule est sélectionnée alors qu'elle ne devrait pas l'être mais que tous les autres pixels appartenant à cette cellule ont une valeur signifiant qu'elle n'est pas sélectionnée, la valeur moyenne des pixels de cette cellule a une plus grande probabilité de signifier que la cellule n'est pas sélectionnée.

Elle permet aussi l'utilisation des valeurs de gris afin de représenter la probabilité qu'un pixel soit sélectionné. Pour utiliser les valeurs de gris, chaque pixel doit pouvoir contenir une valeur située dans l'intervalle de valeurs $[0, 255]$, où 0 est noir et 255 est blanc. Dans ce cas, si la moyenne des valeurs des pixels composant la cellule est supérieure ou égale à 127, la cellule du rectangle de sortie est sélectionnée. Sinon, elle n'est pas sélectionnée. La figure 4.3 présente différents exemples d'images de

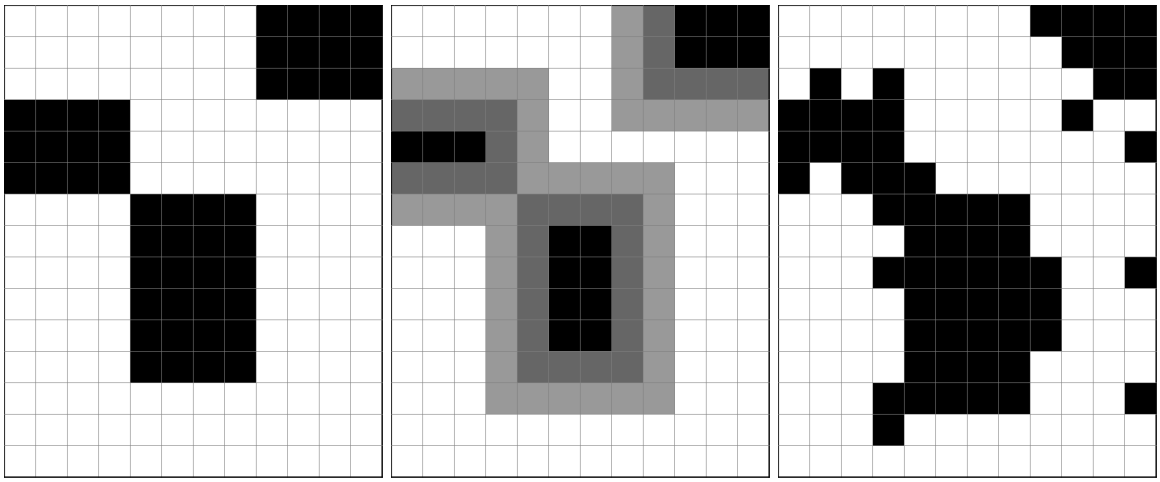
dimensions 12×15 représentant le même serpent maximal d'un rectangle 3×5. Bien que cela ne soit pas illustré, il est possible qu'une image contienne des pixels gris ayant une valeur plus près de la couleur erronée.

FIGURE 4.3 – Images de dimensions 12×15 représentant un serpent maximal d'un rectangle de dimensions 3×5

(a) Image parfaite

(b) Image avec des pixels grisés selon l'incertitude

(c) Image avec des pixels erronés



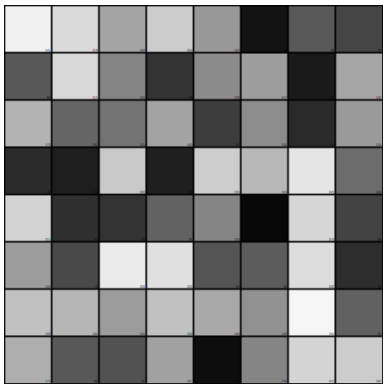
Bruit structuré

L'un des problèmes principaux à résoudre afin de pouvoir générer des rectangles de tailles variées et pouvant être supérieures aux dimensions des rectangles d'entraînement était de savoir comment communiquer les dimensions des cellules, et donc du rectangle, aux réseaux neuronaux. Une solution potentielle à ce problème est l'utilisation de bruit structuré. Habituellement, les images de bruit utilisées sont constituées de pixels ayant des valeurs aléatoire. Ainsi, un bruit régulier dans le projet actuel est une image en noir et blanc où tous les pixels ont une valeur de gris aléatoire entre 0 et 255 inclusivement. Le bruit structuré fonctionne de façon similaire mais les pixels sont regroupés en cellules avant de se voir attribuer une valeur de gris aléatoire commmune. L'hypothèse étant que le générateur pourrait être capable d'apprendre à détecter les

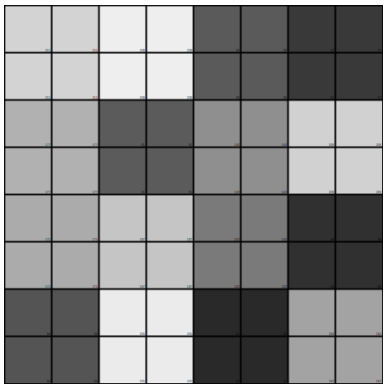
contours de chaque cellule et à les respecter lors de la génération. La figure 4.4 montre la différence entre un bruit régulier et un bruit structuré. Il est important de noter qu'il n'existe aucun lien entre les valeurs de gris des figures 4.4a et 4.4b. Ces images sont générées de façon indépendante et seulement un type de bruit est utilisé à la fois.

FIGURE 4.4 – Exemples de bruit régulier et de bruit structuré dans une image de 8×8 pixels

(a) Bruit régulier entièrement aléatoire



(b) Bruit structuré pour représenter un carré de 4×4 cellules



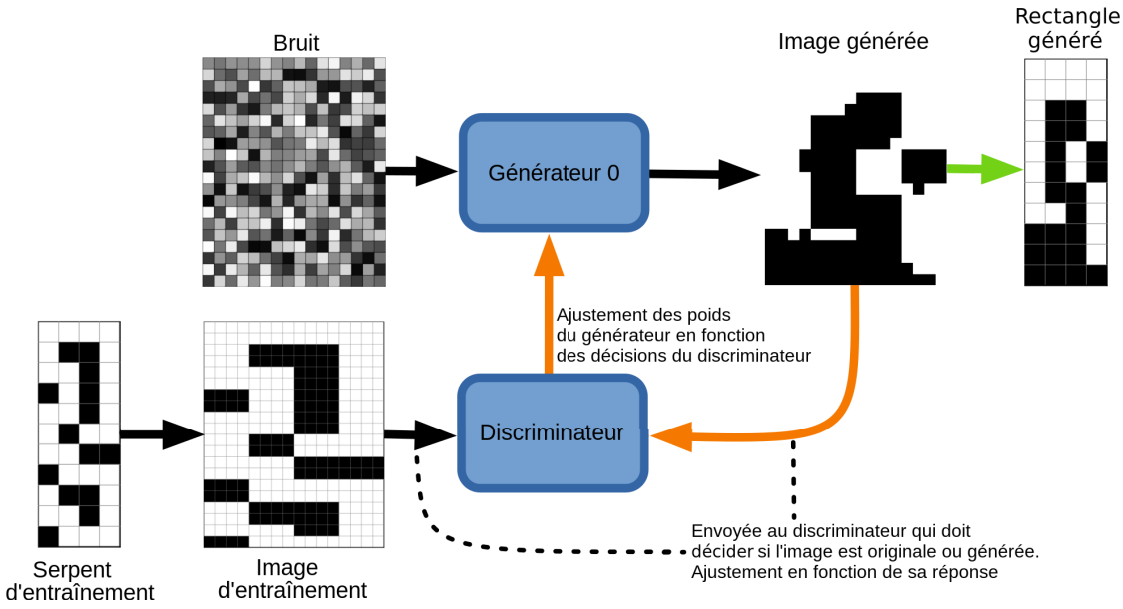
4.3.3 Modèles et configurations

Au cours de ce travail, plusieurs modèles et configurations de GANs ont été testés afin de déterminer l'influence des différents paramètres et pour tenter d'atteindre l'objectif principal, soit de produire des serpents maximaux pour des rectangles plus grands que les rectangles d'entraînement.

La figure 4.5 présente un schéma du fonctionnement général du modèle le plus simple. Une image de bruit est donnée en entrée au générateur qui produit une image représentant un rectangle devant contenir un serpent maximal appartenant à la taille de rectangle demandée à l'aide du bruit structuré et/ou des entrées conditionnelles. Lors de l'entraînement du modèle (flèches noires et oranges de la figure 4.5), cette image est donnée en entrée au discriminateur qui tente de déterminer s'il s'agit d'une

image générée et sa réponse est ensuite utilisée afin de faire l'ajustement des poids de ses neurones et de ceux du générateur. Pour chacune des images générées, le discriminateur reçoit aussi un serpent maximal de la dimension demandée ayant été converti en image afin de s'entraîner à les reconnaître en ajustant ses poids en fonction de la validité de ses réponses. En dehors de l'entraînement, il est possible de n'utiliser que le générateur avec une image de bruit afin de prédire une image et la convertir au format de rectangle désiré (flèche verte de la figure 4.5). Que ce soit lors de l'entraînement ou lors des prédictions, la conversion des rectangles en image et des images en rectangle n'implique pas le GAN. Le réseau n'a donc aucun concept de rectangle ou de cellule puisqu'il ne travaille qu'avec des images de dimensions fixes données en pixels. Il doit apprendre ces concepts tout en apprenant les patrons géométriques des serpents.

FIGURE 4.5 – Schéma complet du GAN utilisant le générateur 0 sans bruit structuré



Dans tous les cas, l'architecture du discriminateur est demeurée constante. Ce discriminateur est composé de deux couches d'entrée, soit une pour l'image et l'autre pour la taille du rectangle désiré. L'entrée de la taille du rectangle est ensuite passée dans une couche dense et restructurée au format de l'image avant de lui être concaténée. Le tout passe ensuite par trois couches convolutionnelles à deux dimensions utilisant la fonction d'activation *LeakyReLU* [48] et un *dropout* [49] de 30%. La fonction d'ac-

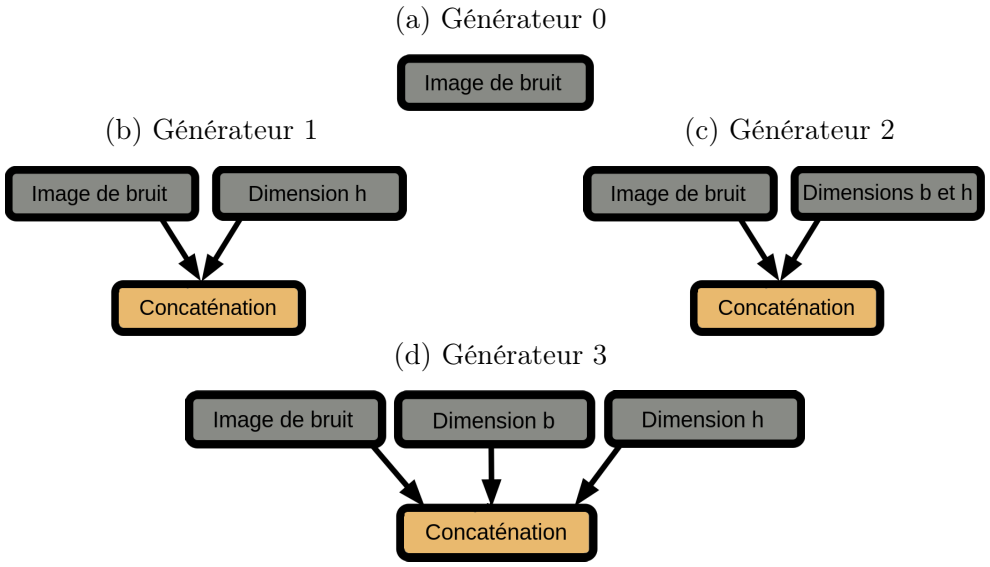
tivation *LeakyReLU* est utilisée puisqu'elle permet un minuscule gradient lorsque le neurone n'est pas activé, ce qui rend le modèle plus robuste lors de l'optimisation. Le *dropout* effectue l'ommission d'un certain pourcentage des neurones du réseaux de façon aléatoire, ce qui réduit les chances de sur-ajustement. Sa valeur de *dropout* a été choisie de façon arbitraire et son impact n'a pas été exploré dans le cadre de ce travail. Pour finir, le résultat est applati en une seule dimension et est passé dans une couche dense de sortie déterminant si l'image d'entrée provient du générateur ou de l'ensemble d'entraînement. Un schéma du discriminateur est présenté à la figure C.2 de l'annexe C.

Pour ce qui est du générateur, quatre modèles légèrement différents ont principalement été utilisés. Ces modèles sont nommés « générateur » suivi d'un numéro et ne peuvent pas être utilisés en même temps. Ils possèdent tous la couche d'entrée correspondante à l'image de bruit, mais les générateurs 1, 2 et 3 sont conditionnels et possèdent donc une ou deux couches d'entrée supplémentaires dont les entrées sont concaténées avec celles de la première couche. Les différents générateurs et leurs couches d'entrée en rapport avec le générateur 0 sont illustrés à la figure 4.6 et décrits dans la liste suivante :

- Le générateur 0 (GEN0) est le générateur le plus simple. Il ne prend comme entrée que l'image de bruit ;
- Le générateur 1 (GEN1) utilise une deuxième couche d'entrée permettant de donner une seule des dimensions du rectangle, soit sa hauteur. Il peut donc uniquement être utilisé lorsque la largeur du rectangle est fixe ;
- Le générateur 2 (GEN2) utilise une deuxième couche d'entrée permettant de donner les deux dimensions du rectangle sous la forme $[(b_1, h_1), (b_2, h_2), (b_3, h_3)...]$;
- Le générateur 3 (GEN3) utilise deux couches d'entrée supplémentaires recevant chacune une dimension du rectangle. La première reçoit la base du rectangle (b) et l'autre sa hauteur (h), donc sous la forme $[b_1, b_2, b_3, ...]$ et $[h_1, h_2, h_3, ...]$.

La concaténation est effectuée de gauche à droite dans l'ordre des entrées telles qu'elles sont présentées dans ces descriptions et dans les schémas de la figure 4.6.

FIGURE 4.6 – Schémas des différences entre les entrées des générateurs



Par la suite, comme le montre le schéma présenté à la figure C.1 de l'annexe C, ces entrées sont passées dans une couche dense suivie de trois couches convolutionnelles transposées. Toutes ces couches sont normalisées et utilisent la fonction d'activation *LeakyReLU*. La couche de sortie est une autre couche convolutionnelle transposée retournant l'image où chaque pixel est encodé en vecteur *one-hot* contenant 256 valeurs continues. Un vecteur *one-hot* est une suite de valeurs continues entre 0 et 1 où la position de la valeur continue la plus élévée correspond à la valeur discrète encodée. Dans ce cas-ci, le vecteur possède 256 valeurs continues permettant de représenter l'ensemble des valeurs discrètes entre 0 et 255, qui sont les couleurs possibles d'un pixel. L'équation 4.1 montre un exemple de vecteur *one-hot* utilisant quatre valeurs continues permettant de représenter une valeur discrète entre 0 et 3 inclusivement. Dans cet exemple, le troisième élément est le plus élevé donc le décodage donnera la valeur discrète 2.

$$[0.16, 0.45, 0.89, 0.64] \Rightarrow 2 \tag{4.1}$$

Perte sémantique

Afin de tenter au maximum de pousser le générateur à représenter la bonne taille de rectangle avec l'image produite, une perte sémantique [50] peut être ajoutée à la perte de base du générateur. Une perte sémantique est une façon supplémentaire d'influencer le comportement du réseau neuronal lors de son entraînement. Elle offre l'avantage de pouvoir être calculée à partir de n'importe quel critère quantifiable auquel le réseau devrait accorder une importance. L'inconvénient majeur de cette méthode est qu'elle nécessite une quantité de calcul supplémentaire considérable ne pouvant pas être effectué sur une carte graphique. Cela augmente de façon significative le temps d'entraînement et rend peu pratique son utilisation dans le cadre du développement itératif de l'architecture d'un réseau neuronal, de ses paramètres et de l'importance à apporter à cette perte. Dans ce travail, la perte sémantique a été appliquée au générateur en l'additionnant à sa perte d'origine. Pour réduire le temps d'entraînement supplémentaire causé par le calcul de la perte sémantique, un paramètre a été introduit afin de déterminer le nombre d'échantillons à utiliser pour ce calcul à chaque lot d'entraînement. Deux façons de calculer la perte sémantique ont été utilisées :

— **SEM1** est calculé à l'aide de la perte d'entropie croisée entre les dimensions du rectangle désiré et de l'estimation des dimensions du rectangle représenté dans l'image générée. L'estimation des dimensions du rectangle représenté est faite pour chaque dimension en établissant l'intervalle régulier de changement d'état (sélectionnée ou non) entre deux pixels selon la fréquence de ce changement d'état. Un exemple complet de ce calcul est présenté à l'annexe D.
— **SEM2** est la moyenne sur toutes les cellules des écart-types des valeurs des pixels de chaque cellule du rectangle désiré.

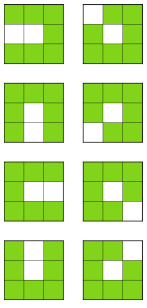
Configurations des modèles

Pour améliorer les performances du GAN, de nombreux modèles ayant des configurations légèrement différentes ont été entraînés et testés. Dans tous les cas, les données d'entraînement sont mélangées aléatoirement et l'entraînement est fait par lot de 32 images. Les différences entre les configurations se situent dans les paramètres suivants :

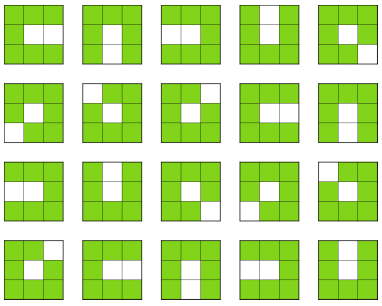
- Les dimensions minimales et maximales des rectangles d'entraînement ;
- Les dimensions de l'image produite et utilisée par le réseau ;
- L'utilisation du bruit structuré ou d'un bruit aléatoire à chaque pixel ;
- Le nombre minimal de serpents pour chaque dimension de rectangle. Ainsi, si le nombre de serpents maximaux existant pour un rectangle $b \times h$ est inférieur à ce seuil, chaque serpent sera dupliqué jusqu'à ce que le seuil soit atteint. La duplication ne dépassera jamais ce seuil. La figure 4.7 présente les deux effets possibles du seuil minimal du nombre de serpents par dimensions de rectangle ;
- La taille de l'échantillonnage utilisé pour calculer la perte sémantique à chaque lot d'entraînement.

FIGURE 4.7 – Exemples de l'effet du seuil minimal du nombre de serpents par dimensions de rectangle

(a) Avec un seuil minimal inférieur ou égal au nombre de serpents à translation près, l'ensemble complet des serpents maximaux est utilisé.



(b) Avec un seuil minimal supérieur au nombre de serpents à translation près, les serpents sont dupliqués, selon l'ordre de l'ensemble initial de serpents, jusqu'à atteindre ce seuil. Dans cet exemple, le seuil minimal est 20 et le nombre de serpents fixes est 8



Le tableau 4.1 présente les valeurs de ces paramètres pour chaque configuration. Dans le reste du document, chaque configuration sera appelée par le numéro présent dans la colonne de gauche du tableau. Ainsi, la configuration 0 est celle qui utilise les rectangles d'entraînement allant de 4×4 à 4×16 cellules, des images 16×20 pixels, le bruit régulier, 0 comme nombre minimum de serpents par dimension de rectangle et 4 échantillons par lot pour calculer la perte sémantique.

TABLE 4.1 – Configurations testées

	Dimensions minimales de rectangles	Dimensions maximales de rectangles	Dimensions de l'image (pixels)	Bruit structuré	Nombre minimal de serpents pour chaque dimension de rectangle	Taille d'échantillonnage pour calculer la perte sémantique
0	4×4	4×16	16×20	Non	0	4
1	4×4	4×16	16×20	Non	100	4
2	4×4	4×16	16×20	Oui	0	4
3	4×4	4×16	16×40	Non	0	4
4	4×4	4×16	16×20	Non	0	32
5	4×4	4×16	16×20	Oui	0	32
6	4×4	4×16	16×20	Oui	0	16
7	4×4	4×16	16×20	Non	0	16
8	4×4	4×16	16×20	Non	0	8
9	4×4	4×16	16×20	Oui	0	8
10	4×4	4×16	16×20	Oui	1000	4
11	4×4	4×16	16×20	Non	1000	4
12	4×4	4×16	16×40	Non	1000	4
13	4×4	4×16	16×40	Oui	1000	4
14	4×4	7×13	16×20	Non	500	4
15	4×4	7×13	16×20	Oui	500	4
16	3×3	3×16	16×20	Non	1000	4
17	5×5	5×16	16×20	Non	1000	4

Afin de réduire la taille des tableaux et des graphiques, les modèles sont identifiés par trois chiffres représentant respectivement le numéro de configuration (défini à la page 55 accompagnée du tableau 4.1), le numéro du générateur (défini à la page 52) et le numéro de la perte sémantique (défini à la page 54) où 0 est utilisé lorsqu'aucune perte sémantique n'est appliquée. Ainsi, le modèle 2-0-1 est le modèle utilisant la configuration 2, le générateur 0 et la SEM1. Le chiffre correspondant à la configuration peut être omis lorsqu'elle est précisée explicitement.

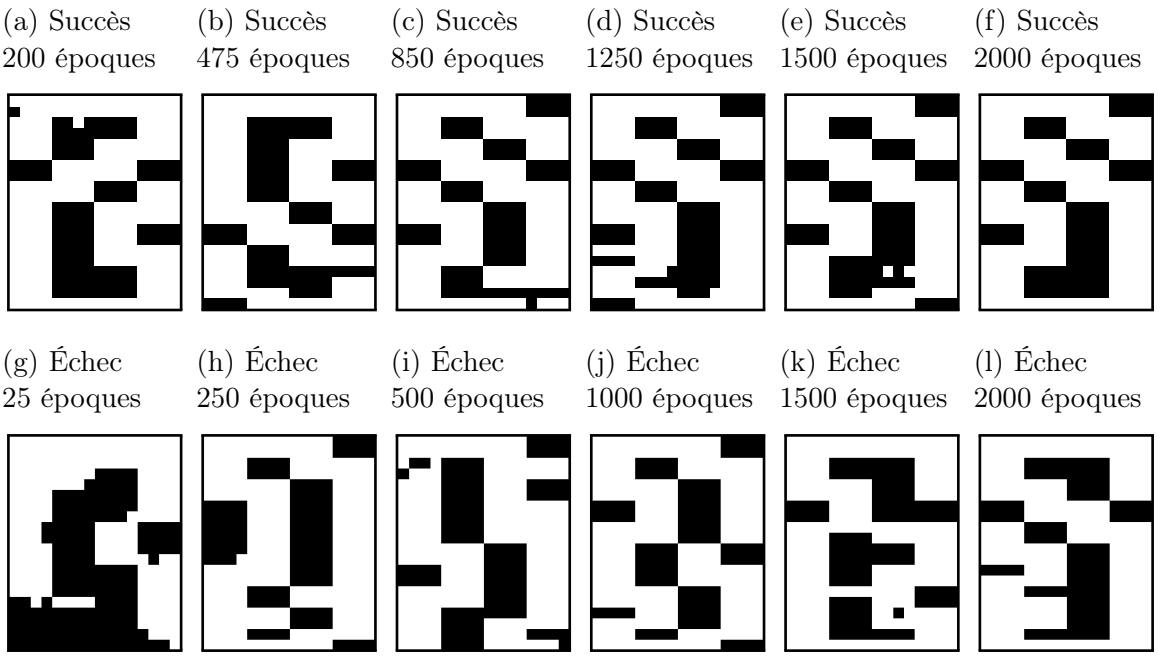
4.3.4 Analyse des résultats

Au cours de ce projet, les modèles ont été entraînés sur 2000 époques, sauf pour exposer ou découvrir un phénomène ou une explication. Une époque correspond à un cycle d'entraînement sur l'ensemble complet de données d'entraînement. Afin de comparer les perfomances des modèles lors des entraînements, le pourcentage de validation est utilisé. Tous les pourcentages de validation ont été obtenus en validant avec 1000 rectangles de dimensions aléatoires et la validation est effectuée à chaque tranche de 25 époques. La validation est effectuée à partir de la sortie du générateur ayant été convertie en un rectangle aux dimensions désirées. Ce rectangle est ensuite comparé à tous les serpents maximaux de cette taille de rectangle et s'il correspond à l'un d'eux, l'image est considéré comme un succès représentant un serpent maximal de la dimension souhaitée. Ainsi, le processus de validation n'implique pas le discriminateur et est entièrement indépendant du modèle. La figure 4.8 montre des exemples d'images générées étant reconnues comme des succès ou des échecs après la conversion d'image à rectangle.

Limitation de la conversion des rectangles en images

À première vue, l'image de la figure 4.8j semble être correcte mais ne l'est pas puisqu'elle représente un serpent maximal d'un rectangle 4×11. De plus, en regardant l'ensemble des images de la figure 4.8, il est possible de s'apercevoir que l'un des défauts de cette méthode est qu'une image représentant un serpent appartenant à un rectangle de dimensions différentes peut être reconnue comme un succès lors de la validation. En effet, la majorité des images de succès présentées ici représentent un rectangle 4×11 alors que le rectangle désiré était de dimensions 4×10. La figure 4.9 présente un exemple d'image pouvant être convertie en un serpent maximal appartenant à un rectangle 4×10 ou être convertie en un serpent maximal appartenant à un rectangle

FIGURE 4.8 – Exemples d'images de rectangles 4×10 générées avec et sans succès lors de la validation après un certain nombre d'époques



4×11.

En observant la figure 4.9, il est possible de constater que, pour un rectangle 4×11, les deux dernières lignes de cellules ont une hauteur d'un seul pixel. Bien qu'il semble raisonnable de penser que l'utilisation d'images de plus grandes dimensions éliminerait la possibilité qu'une image de serpent maximal soit valide pour plus d'une dimension de rectangles, ce n'est pas le cas comme le montrent les tableaux 4.2 et 4.3 présentant les pourcentages des serpents maximaux des rectangles de dimensions 4×4 à 4×16 qui produisent des serpents maximaux pour un rectangle de dimensions différentes à la suite des conversions en images 16×20 et 16×64. Effectivement, comparativement aux images 16×20, l'utilisation d'images 16×64 diminue le nombre d'images de serpents de dimensions spécifiques étant valides pour certaines dimensions mais peut augmenter le pourcentage pour d'autres dimensions.

TABLE 4.2 – Pourcentages des serpents maximaux de chaque taille de rectangle donnant un serpent maximal suite à la conversion en images de dimensions 16×20 et à la reconversion en rectangles

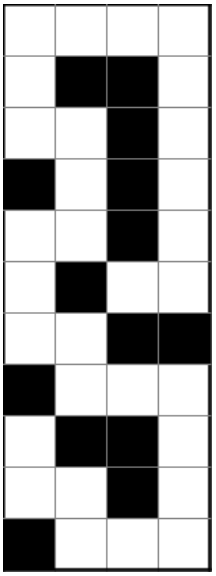
		Sortie												
		4x4	**4x5**	**4x6**	**4x7**	**4x8**	**4x9**	**4x10**	**4x11**	**4x12**	**4x13**	**4x14**	**4x15**	**4x16**
Entrée	**4x4**	100	0	0	0	0	0	0	0	0	0	0	0	0
	4x5	46,15	100	0	0	0	0	0	0	0	0	0	0	0
	4x6	0	0	100	0	0	0	0	0	0	0	0	0	0
	4x7	0	0	0	100	0	0	0	0	0	0	0	0	0
	4x8	0	0	0	10,53	100	0	0	0	0	0	0	0	0
	4x9	0	0	0	0	0	100	0	0	0	0	0	0	0
	4x10	0	0	0	0	0	0	100	0	0	0	0	0	0
	4x11	0	0	0	0	0	0	15,48	100	0	0	0	0	0
	4x12	0	0	0	0	0	0	0	0	100	0	0	0	0
	4x13	0	0	0	0	0	0	0	0	0	100	75	0	0
	4x14	0	0	0	0	0	0	0	0	0	4,23	100	0	0
	4x15	0	0	0	0	0	0	0	0	0	0	0	100	0
	4x16	0	0	0	0	0	0	0	0	0	0	0	0	100

TABLE 4.3 – Pourcentages des serpents maximaux de chaque taille de rectangle donnant un serpent maximal suite à la conversion en images de dimensions 16×64 et à la reconversion en rectangles

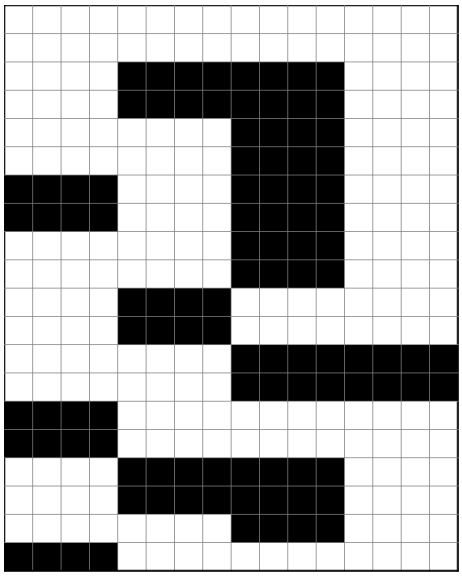
		Sortie												
		4x4	**4x5**	**4x6**	**4x7**	**4x8**	**4x9**	**4x10**	**4x11**	**4x12**	**4x13**	**4x14**	**4x15**	**4x16**
Entrée	**4x4**	100	0	0	0	0	0	0	0	0	0	0	0	0
	4x5	46,15	100	0	0	0	0	0	0	0	0	0	0	0
	4x6	0	0	100	0	0	0	0	0	0	0	0	0	0
	4x7	0	0	0	100	0	0	0	0	0	0	0	0	0
	4x8	0	0	1,32	7,89	100	0	0	0	0	0	0	0	0
	4x9	0	0	0	0	0	100	0	0	0	0	0	0	0
	4x10	0	0	0	0	0	0	100	0	0	0	0	0	0
	4x11	0	0	0	0	0	0	5,95	100	0	0	0	0	0
	4x12	0	0	0	0	0	0	0	0	100	0	0	0	0
	4x13	0	0	0	0	0	0	0	0	0	100	0	0	0
	4x14	0	0	0	0	0	0	0	0	0	4,93	100	0	0
	4x15	0	0	0	0	0	0	0	0	0	0	0	100	0
	4x16	0	0	0	0	0	0	0	0	0	0	0	0	100

FIGURE 4.9 – Exemple de serpent maximal du rectangle 4×11 qui, après la conversion en image 16×20, donne un serpent maximal du rectangle 4×10 après avoir été reconverti en cette dimension

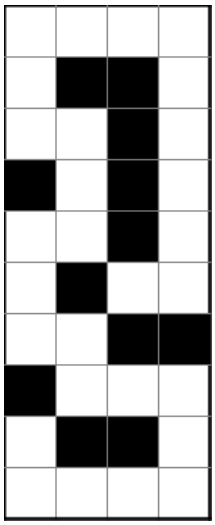
(a) Serpent maximal d'un rectangle 4×11



(b) Après la conversion de la figure 4.9a en image 16×20



(c) Après la conversion de l'image de la figure 4.9b en rectangle de 4×10



Dimensions de rectangles d'entraînement à faible validation

Pour tous les modèles, certaines dimensions spécifiques de rectangles ont des pourcentages de validation à 0% et d'autres ont des pourcentages de validation très bas relativement aux autres dimensions de rectangles. Cela peut être observé dans le tableau E.1 présenté dans l'annexe E, qui montre aussi que les dimensions 4×7, 4×12 et 4×16 sont particulièrement problématiques parmi tous les modèles entraînés avec les rectangles de dimensions 4×4 à 4×16. Étant donné que même les modèles utilisant 1000 comme nombre minimal de serpents pour chaque dimension de rectangle, il est possible de conclure que la faiblesse des modèles face à ces dimensions n'est pas due à un mauvais équilibre dans l'ensemble de données.

L'hypothèse suivante est que ces résultats négatifs sont peut-être dûs à la conversion entre les images de dimensions fixes et les rectangles représentés. Cette hypothèse pourrait aussi expliquer le fait que les modèles génèrent, pour certaines dimensions

de rectangles, des images de serpents maximaux appartenant à un rectangle de dimensions légèrement différentes de celles désirées. Afin de vérifier cette hypothèse, le tableau 4.2 présente le pourcentage des serpents maximaux étant valides après avoir été convertis en images de 16×20 pixels et reconvertis en un rectangle de dimensions différentes. Il est possible d'y voir que certains serpents maximaux sont valides pour plus d'une dimension de rectangles après ces transformations. Malheureusement, cela n'explique pas pourquoi les dimensions 4×7, 4×12 et 4×16 produisent d'aussi mauvais résultats puisqu'elles ne contiennent pas ou très peu de serpents où la conversion produit des résultats mixtes. Cependant, le fait que certains serpents maximaux donnent aussi des serpents maximaux pour des dimensions différentes suite aux conversions de rectangle à image et d'image à rectangle (Figure 4.9) peut expliquer pourquoi les exemples de succès présentés dans la figure 4.8 sont des images de serpents de rectangles 4×11 plutôt que des rectangles 4×10.

Une autre des hypothèses permettant d'expliquer ces résultats est la faible diversité des serpents de ces dimensions spécifiques et donc que la duplication des données afin d'en augmenter le nombre ne permet pas de résoudre le problème d'équilibrage des données. Bien que cela semble vrai pour la dimension 4×7 puisqu'il n'existe que 16 serpents maximaux à translation près, ce qui est le plus petit nombre de serpents maximaux pour un rectangle $4 \times h$. Par contre, les rectangles de dimensions 4×12 et 4×16 ont respectivement 64 et 40 serpents maximaux à translation près. Cela est plus élevé que le nombre de serpents maximaux à translation près des dimensions 4×5, 4×6, 4×10 et 4×13, qui ne montrent pas le même problème de performance. La faible diversité des serpents maximaux ne semble donc pas être la cause des mauvaises performances des modèles pour les dimensions 4×7, 4×12 et 4×16. Toutefois, un entraînement a été effectué avec l'ensemble des rectangles de dimensions 3×3 à 3×16 et un autre avec celui des rectangles 5×5 à 5×16. Dans ces deux cas, les seules dimensions ayant de très faibles taux de validation sont les dimensions 3×6 et 5×10 qui possèdent seulement deux serpents maximaux à translation près. Il semble donc que la faible diversité des serpents maximaux soit effectivement une cause de

la diminution de la performance d'apprentissage du réseau, bien que le seuil minimal soit extrêmement bas.

Prédictions pour des rectangles plus grands

Malheureusement, tous les modèles de réseaux neuronaux entraînés lors de ce travail ont échoués à restreindre la taille en pixels des cellules dans les images afin de pouvoir représenter un serpent appartenant à un rectangle plus grand que ceux utilisés lors de l'entraînement. C'est pour cette raison que les pourcentages de validations ont été utilisés pour cette analyse des résultats.

Au cours des tests, le générateur tente de produire un serpent appartenant au rectangle connu (ayant fait partie de l'entraînement) ayant les dimensions le plus près des dimensions demandées. Il est aussi intéressant que plus l'écart entre les dimensions connues et demandées est grand, plus le modèle utilise les teintes de gris dans l'image générée. Cette progression de l'utilisation de pixels gris est facilement visible dans les figures 4.10 et 4.11 qui présentent des images générées par les modèles 12-2-0 et 14-2-0 après un entraînement de 2000 époques et pour des dimensions de rectangle supérieures à celles d'entraînement.

FIGURE 4.10 – Images générées par le modèle 12-2-0 après un entraînement de 2000 époques pour des dimensions de rectangle supérieures à celles d'entraînements

(a) Rectangle 4×17 (b) Rectangle 4×18 (c) Rectangle 4×19 (d) Rectangle 4×20

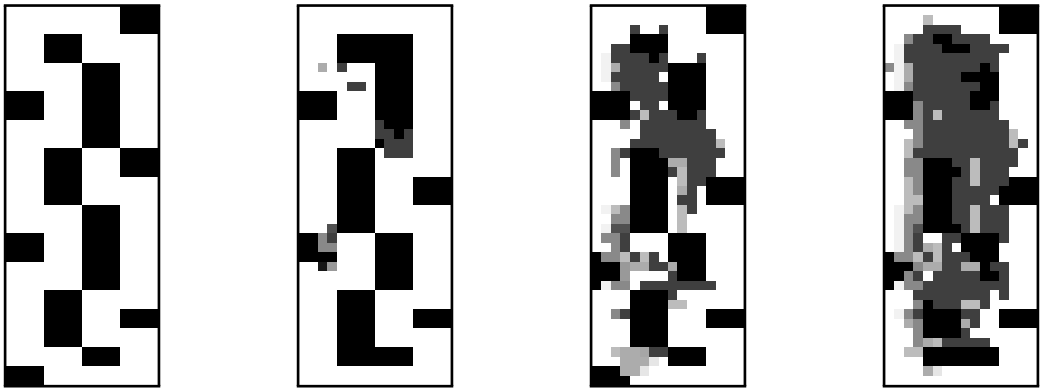
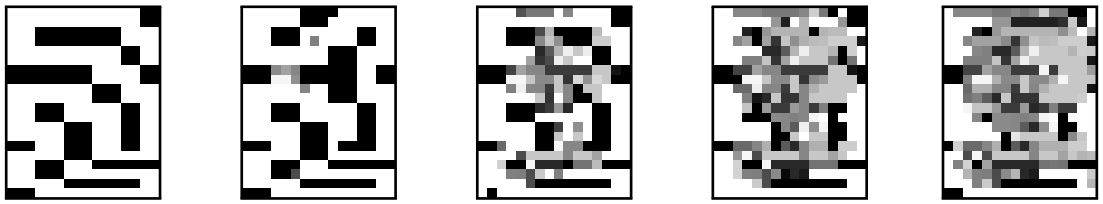


FIGURE 4.11 – Images générées par le modèle 14-2-0 après un entraînement de 2000 époques pour des dimensions de rectangle supérieures à celles d'entraînements

(a) Rectangle 6×14 (b) Rectangle 6×15 (c) Rectangle 6×16 (d) Rectangle 6×17 (e) Rectangle 6×18

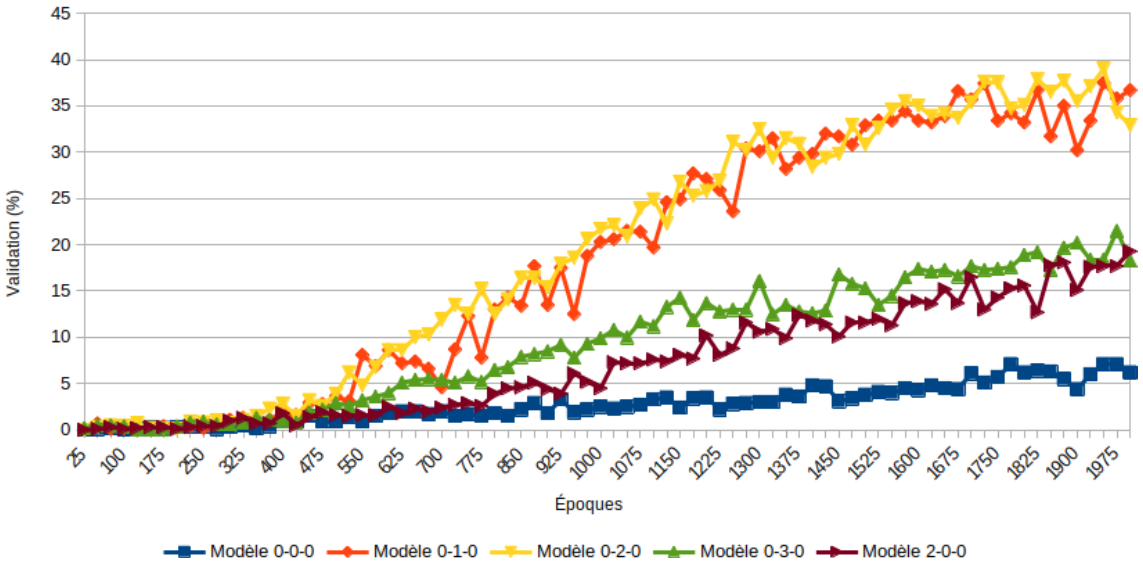


Comparaison des générateurs

En comparant les taux de validation des différents générateurs sans perte sémantique et avec la configuration de base (configuration 0) présentés dans la figure 4.12, où les modèles sont identifiés par leurs numéros de configuration, de générateur et de perte sémantique, il est facile de réaliser que les générateurs 1 et 2 sont de loin les plus performants. Le générateur 2 offre l'avantage de pouvoir être utilisé sur des rectangles où les deux dimensions peuvent être choisies. Il a donc été choisi pour évaluer les impacts des autres paramètres des modèles et des configurations. Le générateur 0 utilisant le bruit structuré est aussi inclus dans cette comparaison puisqu'il s'agit du seul générateur non conditionnel et qu'il pourrait être intéressant de savoir si l'impact de chacun des paramètres sera similaire à l'impact sur un générateur conditionnel, soit

les générateurs ayant au moins une entrée spécifique aux dimensions du rectangle.

FIGURE 4.12 – Comparaison du pourcentage de validation des différents générateurs

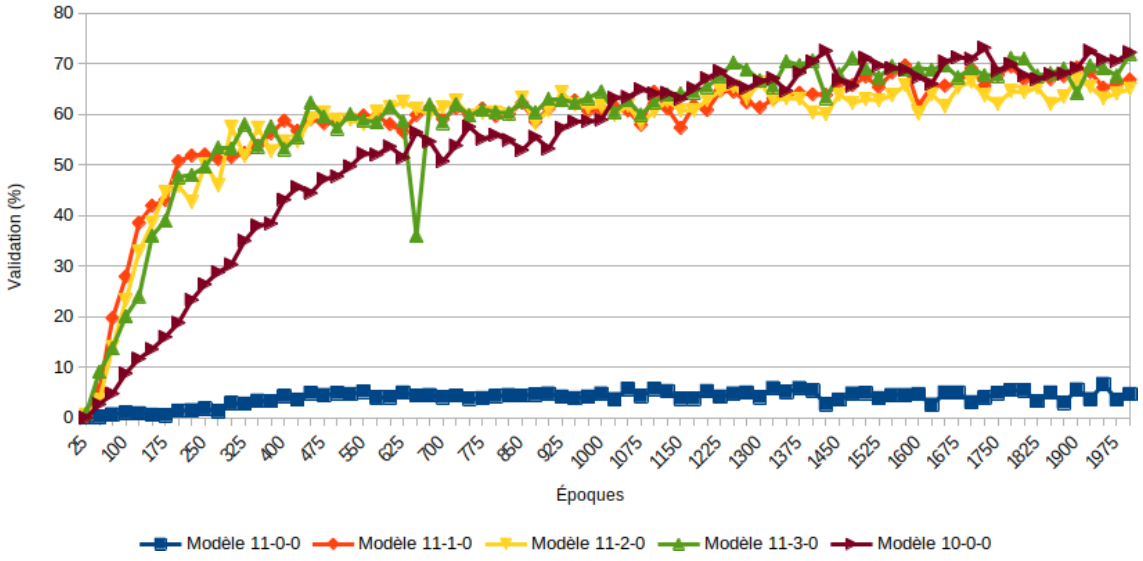


Toutefois, en modifiant le nombre minimum de serpents par dimensions de rectangle, le classement de la performance des différents générateurs est différent. En effet, en observant la figure 4.13 représentant ces nouveaux résultats, il est possible de constater que tous les générateurs, à l'exception du générateur 0 n'utilisant pas le bruit structuré, offrent des performances similaires. De plus, le générateur 0 utilisant le bruit structuré et le générateur 3 se retrouvent alors au sommet du classement des pourcentages de validation, bien que de très peu, ce qui pourrait être différent d'un entraînement à l'autre.

Un autre fait intéressant est que le générateur 0 utilisant le bruit structuré présente une progression plus lente lors de l'entraînement. Cela s'explique fort probablement par le fait que le réseau doit apprendre à reconnaître les tailles représentées dans les bruits structurés plutôt que de se faire fournir les dimensions directement.

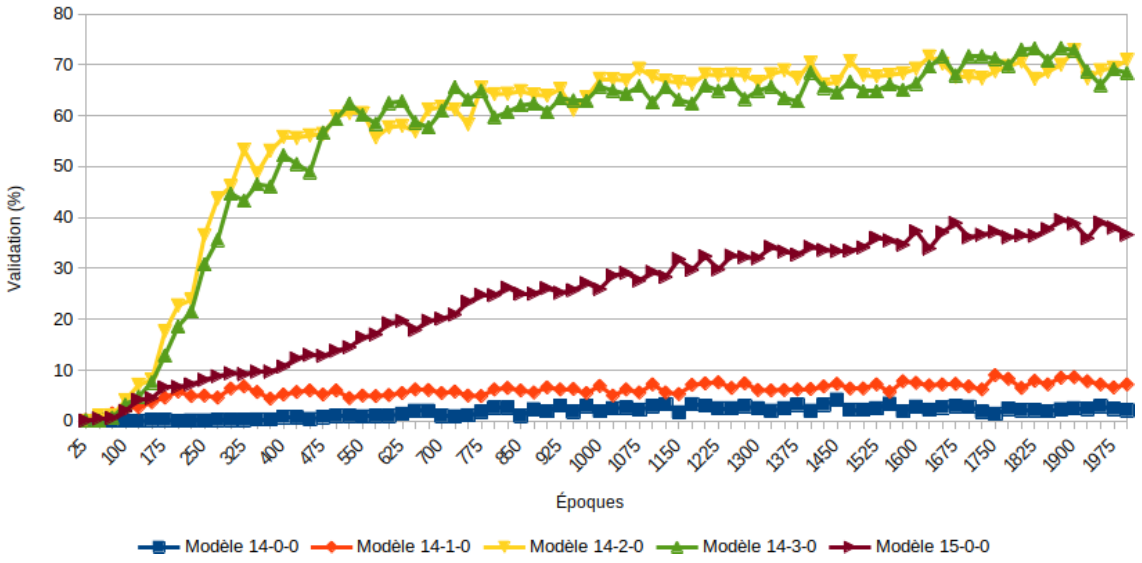
Tel que mentionné précédemment, les générateurs possédant les deux dimensions de rectangles en entrées (GEN2 et GEN3) sont les meilleurs pour un modèle plus général capable de s'ajuster à des rectangles où les deux dimensions sont variables.

FIGURE 4.13 – Comparaison du pourcentage de validation des différents générateurs avec un minimum de 1000 serpents maximaux par dimensions de rectangle



De plus, le générateur 0 utilisant le bruit structuré est aussi capable d'apprendre ce cas d'utilisation, bien que ses performances restent inférieures à celles des modèles conditionnels. La figure 4.14 montre les performances des différents modèles pour les rectangles de dimensions 4×4 à 7×13. Il est aussi possible d'y constater que le générateur 3 (en vert) offre une performance équivalente à celle du générateur 2 (en jaune). Il semble donc que le générateur 3 nécessite une plus grande quantité minimale de données que le générateur 2 afin de pouvoir fonctionner.

FIGURE 4.14 – Comparaison du pourcentage de validation des différents générateurs avec un minimum de 500 serpents maximaux par dimensions de rectangle où les deux dimensions sont variables

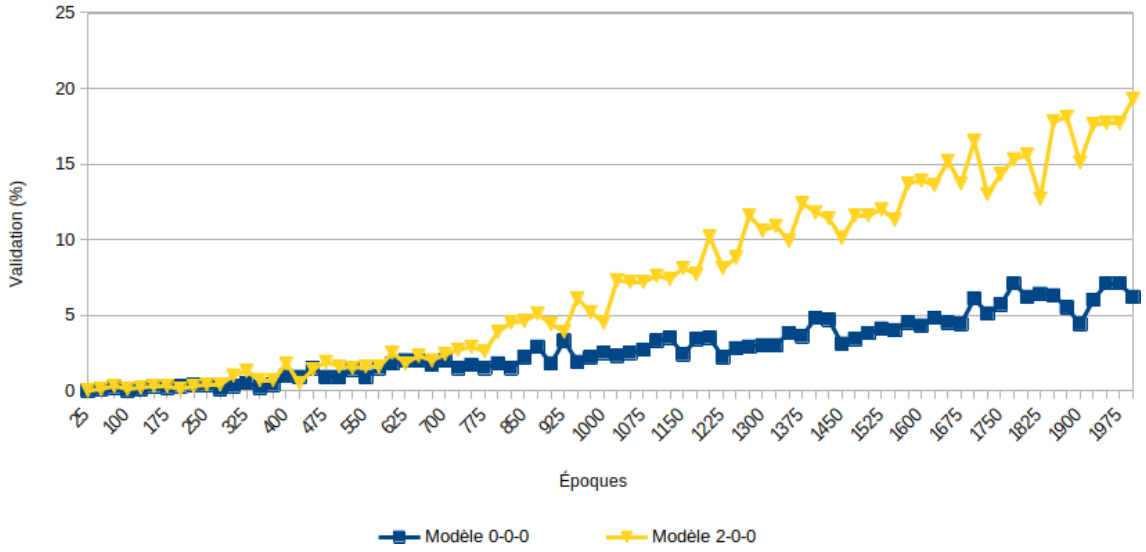


Impact du bruit structuré

L'utilisation du bruit structuré avec le générateur 0, soit le seul générateur ne recevant pas au moins une des dimensions du rectangle en entrée, augmente considérablement ses performances. Cette augmentation de performance peut être observé dans chacun des entraînements, indépendamment des autres paramètres des configurations. Par exemple, après un entraînement de 2000 époques, le modèle n'utilisant pas le bruit structuré est incapable d'identifier correctement ce qu'il doit générer alors que celui utilisant le bruit structuré atteint un maximum de 20% lors de la validation, comme le montre la figure 4.15.

En comparant les résultats du générateur 0 utilisant le bruit structuré à ceux obtenus par les générateurs conditionnels, il est possible de constater une courbe et des taux très similaires. Cela est visible dans la figure 4.16, où les modèles sont identifiés par leurs numéros de configuration, de générateur et de perte sémantique. L'une des hypothèses pouvant expliquer ces résultats serait que l'utilisation du bruit structuré est suffisante pour permettre aux modèles de générer la bonne dimension

FIGURE 4.15 – Comparaison du pourcentage de validation du générateur 0 avec et sans l'utilisation du bruit structuré



de rectangle et que l'utilisation d'un générateur conditionnel recevant au moins une dimension du rectangle en entrée n'offre aucun avantage. Par contre, en comparant ces pourcentages de validation à ceux obtenus par les générateurs 1 et 2 n'utilisant pas le bruit structuré (Figure 4.17), on s'aperçoit que ces derniers sont beaucoup plus performants. Il semblerait donc que l'utilisation du bruit structuré restreigne l'amélioration des générateurs conditionnels lors de l'entraînement.

Une autre constatation intéressante est que le générateur 0 utilisant le bruit structuré converge vers un taux de validation moyen de 35% lors d'un entraînement de 10000 époques, qui est atteint par les générateurs conditionnels n'utilisant pas le bruit structuré après seulement 2000 époques. La comparaison avec le modèle 0-2-0, présenté par la figure 4.18, suggère que l'utilisation d'un générateur conditionnel offre un plus grand potentiel de performance qu'un générateur standard utilisant le bruit structuré. Toutefois, étant donné qu'aucun des modèles n'a réussi à générer une image d'un rectangle plus grand que ceux d'entraînement, il est impossible de conclure qu'un générateur conditionnel serait la meilleure solution pour ce cas d'utilisation.

FIGURE 4.16 – Comparaison du pourcentage de validation pour différents générateurs utilisant le bruit structuré

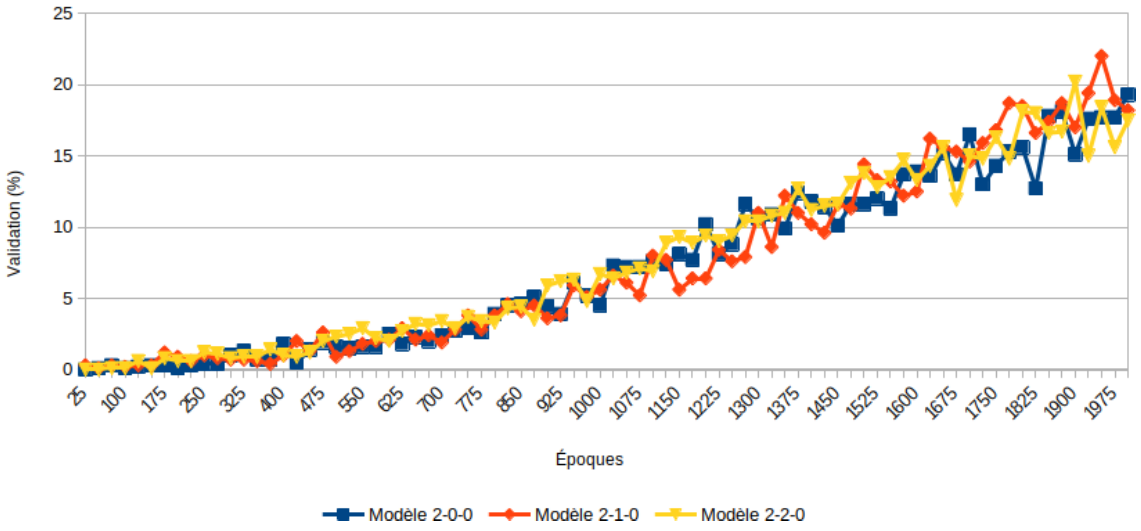


FIGURE 4.17 – Comparaison du pourcentage de validation des générateurs 1 et 2 avec et sans bruit structuré

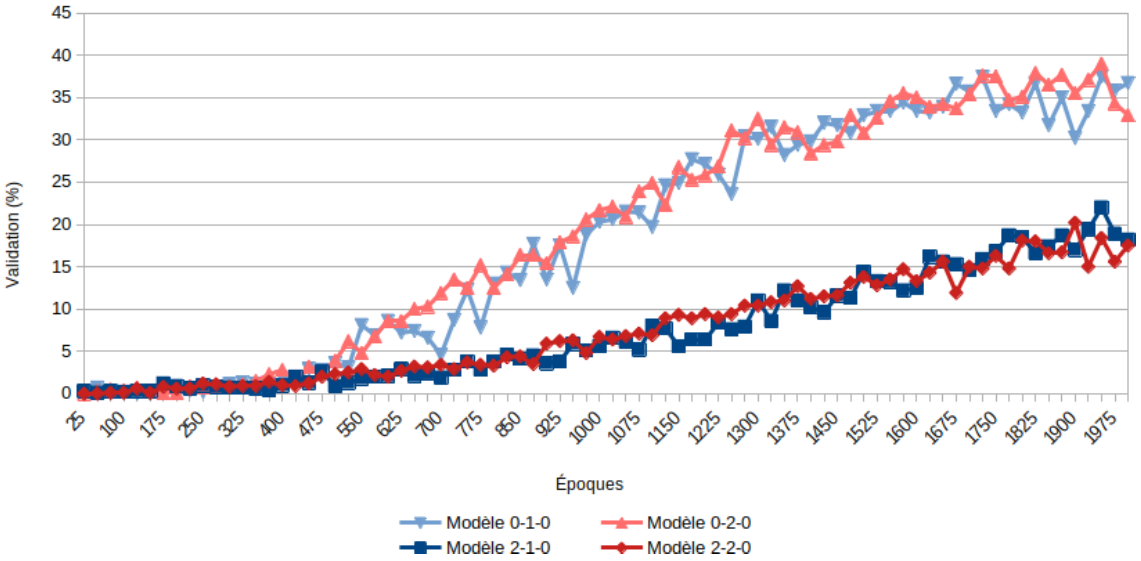
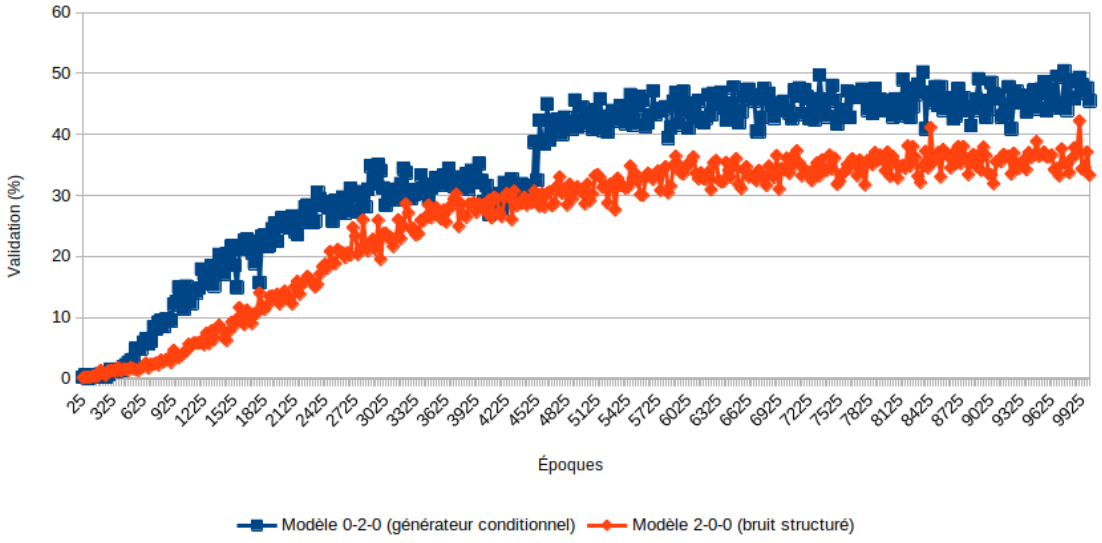


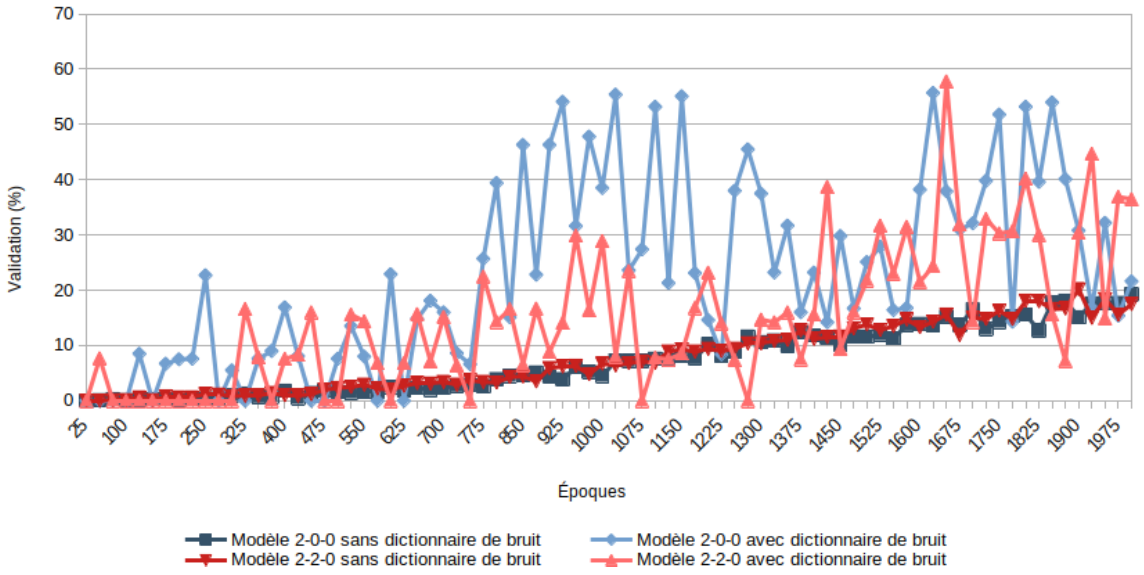
FIGURE 4.18 – Comparaison des pourcentages de validation des modèles 0-2-0 et 2-0-0 sur 10000 époques



Impact de l'utilisation d'un dictionnaire de bruits

Certains tests ont aussi été réalisés en utilisant un dictionnaire de bruits, ce qui signifie que pour chaque dimension de rectangle utilisé, un bruit structuré unique est généré au début de l'entraînement et est réutilisé sur toute sa durée. Les deux modèles utilisés pour cette expérimentation sont les générateurs 0 et 2 sans perte sémantique suivant la configuration 2, donc utilisant le bruit structuré. La figure 4.19 montre l'évolution de la précision de ces modèles avec et sans le dictionnaire de bruits. Il est possible de constater que l'utilisation d'un dictionnaire de bruits produit une précision très erratique. Les images produites par les générateurs utilisant un dictionnaire de bruits montrent aussi une très faible diversité. L'explication la plus probable de cette faible diversité de génération est que les entrées possèdent elles aussi une faible diversité puisqu'elles utilisent le dictionnaire de bruits qui utilise le même bruit pour une dimension spécifique.

FIGURE 4.19 – Comparaison de la précision de deux modèles lors de l'entraînements avec et sans dictionnaire de bruits

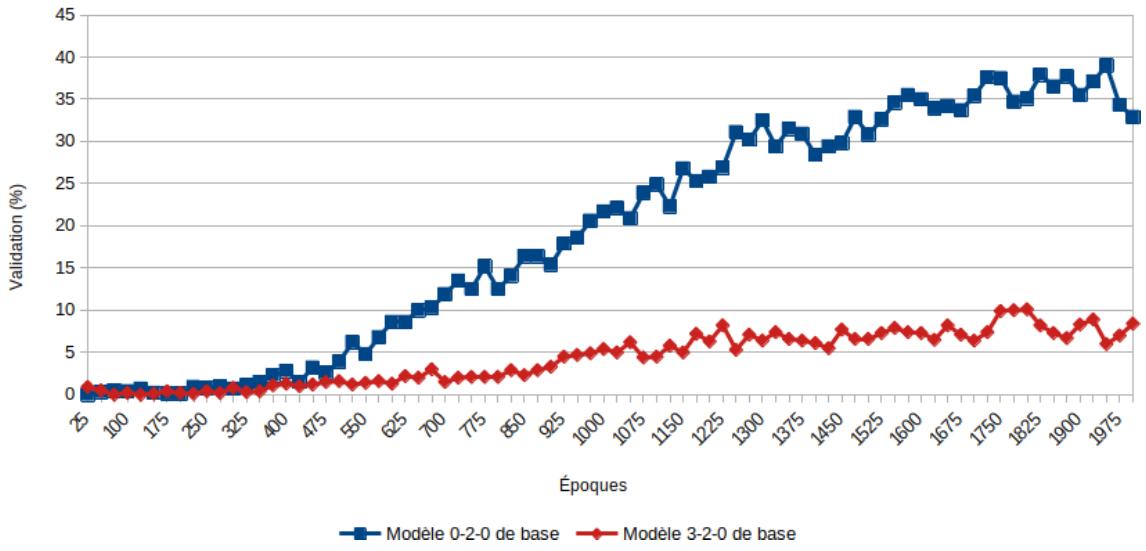


Impact de la taille de l'image

Au fil du développement des expérimentations et des différentes itérations des modèles, l'une des hypothèses était que les modèles seraient plus performants en utilisant de plus grandes dimensions d'images. Cette théorie a été testée en utilisant le générateur 2 sans perte sémantique pour des images de dimensions 16×20 et 16×40 pixels (respectivement en bleu et en rouge dans les figures 4.20, 4.21 et 4.22). En pratique, il semble que le modèle soit plus adapté aux images de petite taille comme le montrent les résultats de la figure 4.20, qui montre une bien meilleure performance en utilisant des images plus petites.

L'une des explications potentielles de ce phénomène est que le modèle ne contient peut-être pas assez de filtres puisque les trois couches convolutionnelles transposées du générateur ne possèdent respectivement que 256, 128 et 128 filtres. Il se pourrait donc que cela ne permette pas au générateur de s'ajuster correctement pour une image contenant le double de pixels. Cette explication a été mise à l'épreuve en modifiant le générateur pour que ces trois couches convolutionnelles transposées aient respecti-

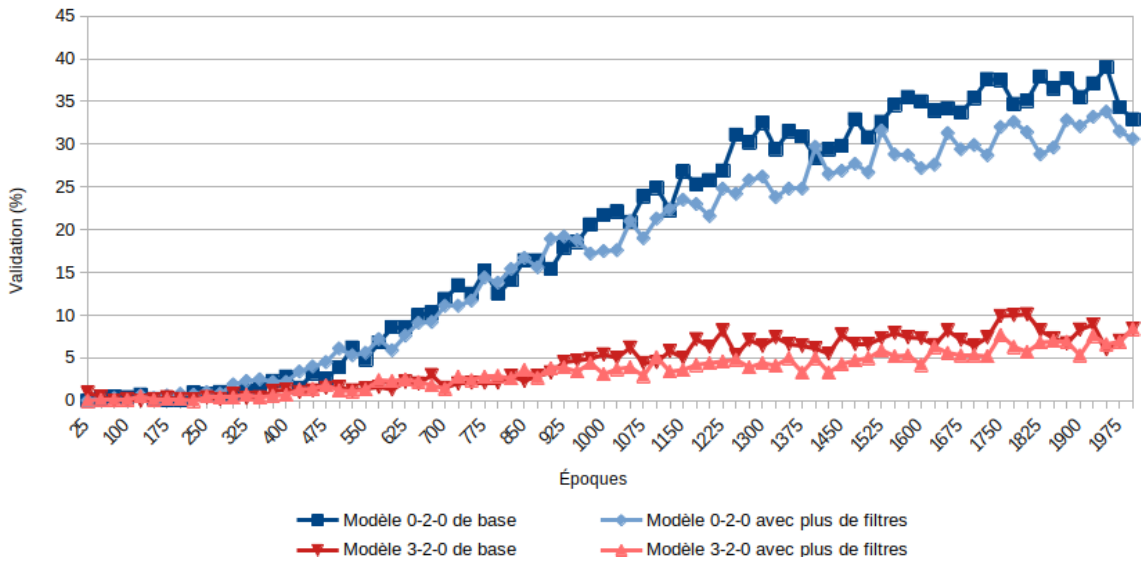
FIGURE 4.20 – Comparaison du pourcentage de validation du générateur 2 utilisant soit des images de dimensions 16×20 (en bleu), soit de 16×40 (en rouge)



vement 1024, 512 et 256 filtres. Toutefois, cela n'a pas résolu le problème comme le montrent les résultats de la figure 4.21. En effet, les résultats sont très similaires et les variations sont similaires à celles pouvant être observées lors de deux entraînements d'un modèle identique. Ces résultats permettent aussi de conclure qu'une augmentation du nombre de filtres des couches existantes du générateur n'a pas d'impact significatif sur sa performance.

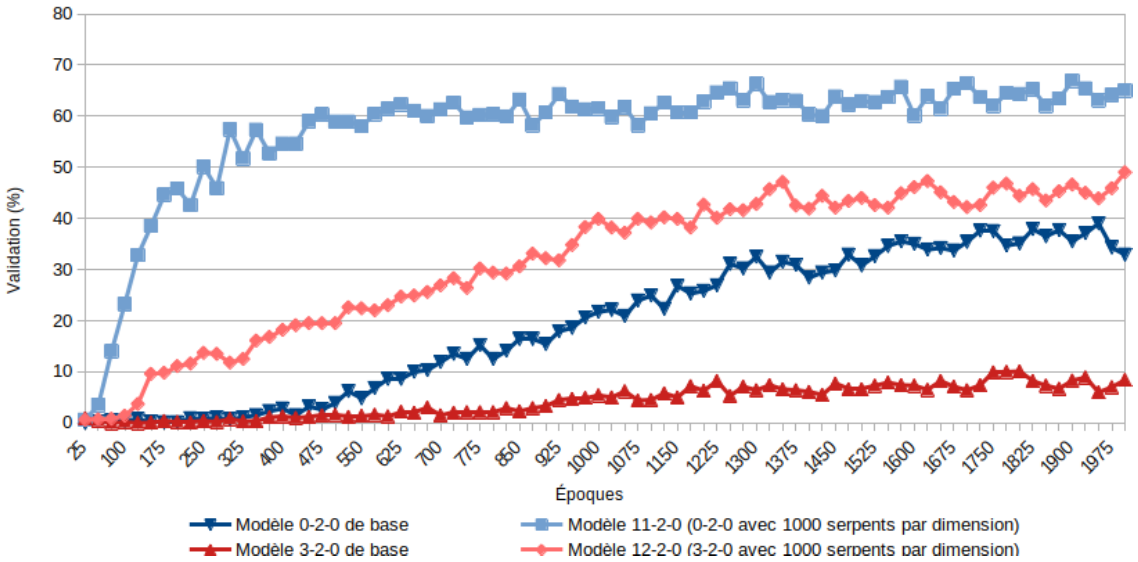
Finalement, en effectuant des entraînements avec un ensemble de données contenant un seuil minumum de 1000 serpents par dimension de rectangles, obtenus par duplication lorsque l'ensemble de serpents à translation près est plus petit que ce seuil, il a été possible de voir que ces modèles sont capables d'apprendre en utilisant une taille d'images de 16×40 pixels. La figure 4.22, où les modèles sont identifiés par leurs numéros de configuration, de générateur et de perte sémantique, montre ces résultats ainsi que l'impact de l'utilisation de plus grandes images sur la vitesse d'apprentissage. L'impact sur la vitesse d'apprentissage était prévisible étant donné l'augmentation du nombre de paramètres. Il est donc possible de conclure que l'utilisation d'images de plus grandes dimensions nécessite un plus grand nombre de données

FIGURE 4.21 – Effet de l'ajout de filtres supplémentaires au générateur. Les dimensions des images utilisées par les modèles en bleu sont 16×20 et celles des modèles en rouge sont 16×40



ainsi qu'un nombre d'époques supplémentaires afin de mieux performer.

FIGURE 4.22 – Effet de l'utilisation de 1000 comme nombre minimal de serpents par dimensions de rectangle. Les dimensions des images utilisées par les modèles en bleu sont 16×20 et celles des modèles en rouge sont 16×40



Impact de la perte sémantique

Étant donné l'impact élevé de l'utilisation de la perte sémantique sur la durée de l'entraînement d'un modèle, les tests initiaux ont été effectués avec un échantillonnage de 4 par lot. Les résultats obtenus et présentés dans la figure 4.23 montrent que dans la majorité des cas, l'application de la perte sémantique produit des résultats similaires ou inférieurs. Par contre, il est possible d'observer l'effet contraire dans le cas du générateur 3, où l'application d'une perte sémantique améliore largement les performances du modèle. Cela pourrait être expliqué par le fait que le générateur 3 est le générateur conditionnel ayant les pires taux de validation sans l'application d'une perte sémantique.

À la vue de ces résultats, il est possible de conclure que l'utilisation d'un échantillonnage de 4 par lot est suffisant pour que l'application de la perte sémantique ait un

impact sur l'entraînement d'un modèle. De plus, la perte sémantique utilisant l'écart-type des pixels d'une même cellule (SEM2) semble supérieure à la fonction SEM1. Ce résultat était attendu puisque le calcul de la fonction SEM2 est beaucoup plus robuste et précis que l'estimation de la taille du rectangle effectuée pour la fonction SEM1.

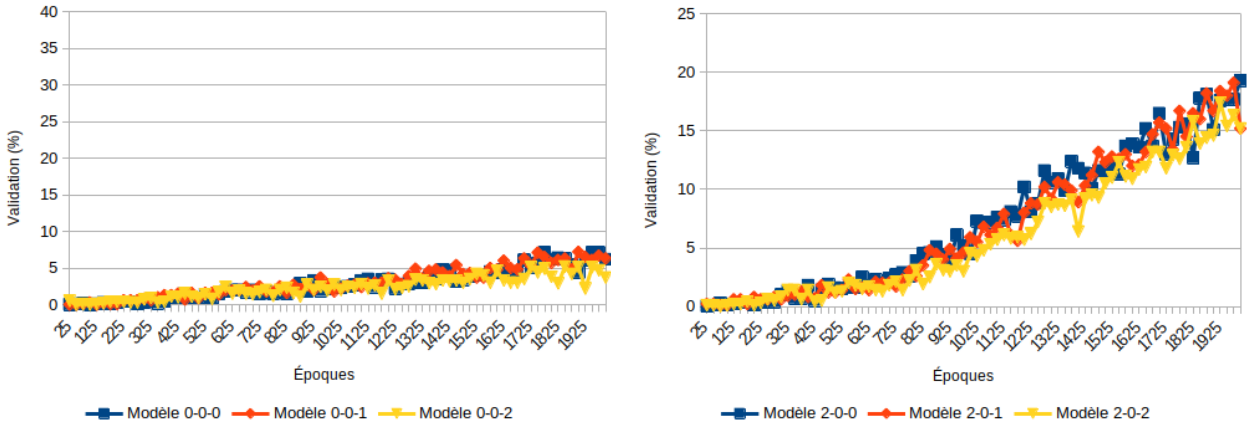
En effectuant d'autres entraînements avec différentes tailles d'échantillonnage pour le calcul de la perte sémantique, les résultats obtenus dans la figure 4.24 montrent que les performances des deux fonctions de perte sémantique sont similaires et donc, qu'en pratique, le calcul de la perte sémantique avec SEM2 n'est pas supérieur à celui avec SEM1. Ces résultats montrent aussi que peu importe la taille d'échantillonnage, l'implémentation actuelle de la perte sémantique n'offre aucun bénéfice aux modèles. Il est cependant important de mentionner qu'en raison du temps computationnel nécessaire, aucun test n'a combiné l'utilisation de la perte sémantique à celle d'un minimum de serpents par dimensions de rectangle. Il se pourrait donc que la perte sémantique soit capable de produire une amélioration avec l'utilisation d'un ensemble d'entraînement contenant le même nombre de serpents pour chaque dimension.

Pour ce qui est de la taille d'échantillonnage utilisée pour le calcul, des entraînements ont été réalisés sur les rectangles allant de 4×4 à 4×16 en utilisant les tailles d'échantillonnage 0, 4, 8, 16 et 32 pour le calcul de la perte sémantique. Suite à ces entraînements, il est intéressant de constater que le classement des taux de validation du générateur 2 est identique pour les deux fonctions de perte. Ce résultat est intéressant puisque ce classement n'indique pas de correlation entre la taille d'échantillonnage utilisée et le pourcentage de validation du modèle. Cela était inattendu et reste un mystère à résoudre.

FIGURE 4.23 – Comparaison des taux de validation des différents générateurs en utilisant un échantillonnage de 4 pour le calcul de la perte sémantique

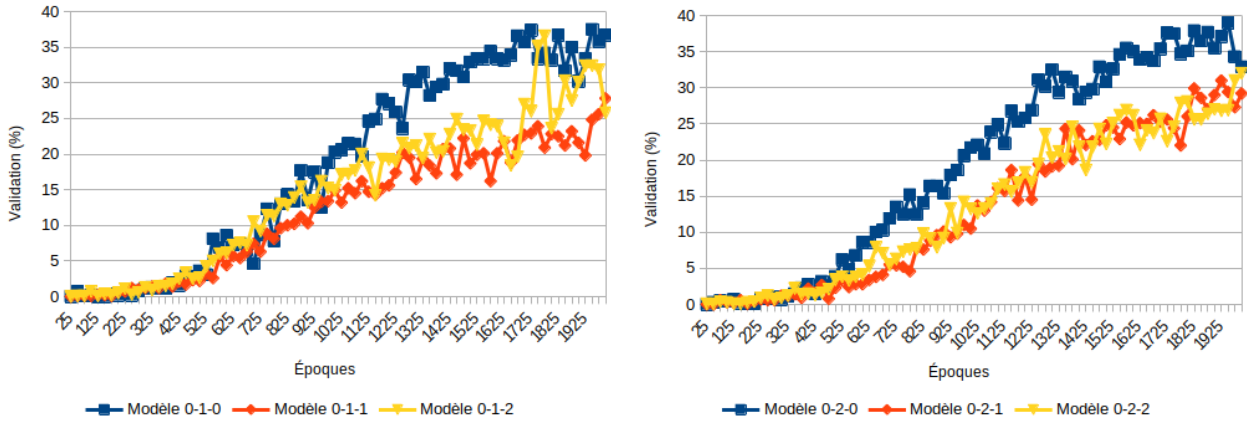
(a) Générateur 0 sans bruit structuré

(b) Générateur 0 avec bruit structuré



(c) Générateur 1

(d) Générateur 2



(e) Générateur 3

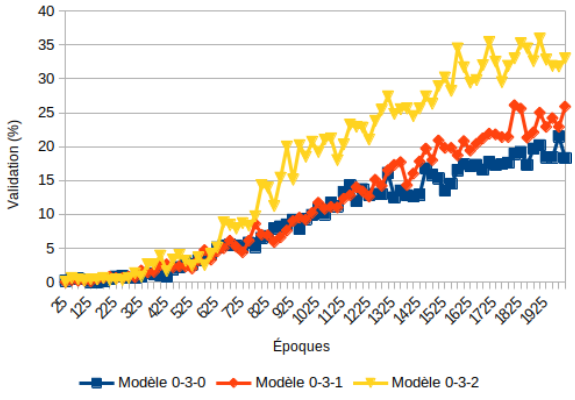
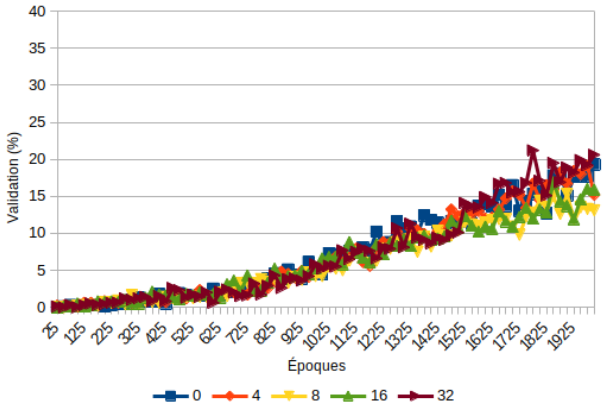
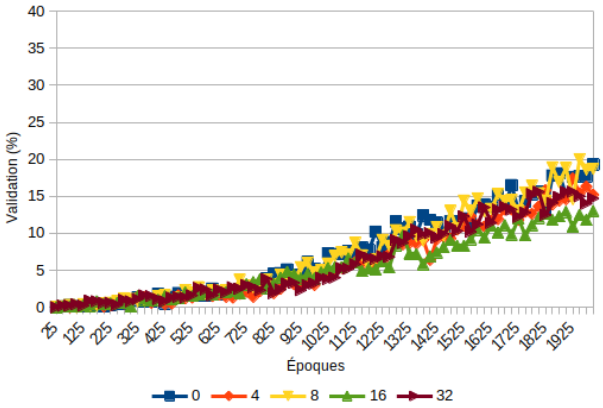


FIGURE 4.24 – Comparaison des taux de validation des modèles 2-0-1, 2-0-2, 0-2-1 et 0-2-2 en utilisant différentes tailles d'échantillonnage pour le calcul de la perte sémantique

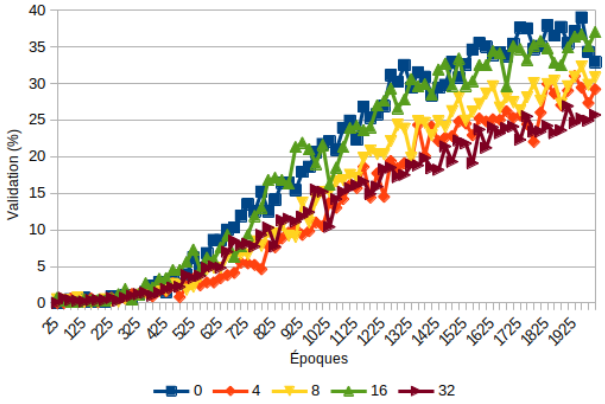
(a) Générateur 0 avec la perte sémantique 1



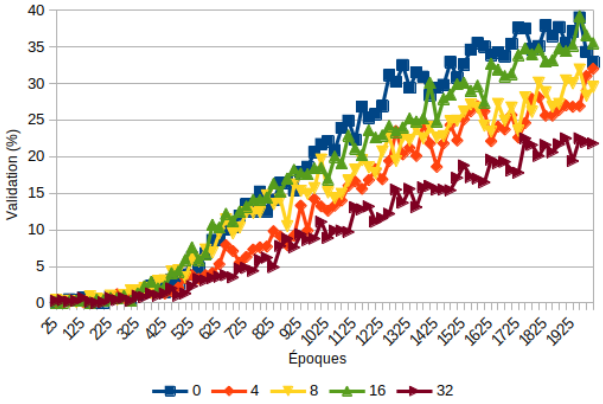
(b) Générateur 0 avec la perte sémantique 2



(c) Générateur 2 avec la perte sémantique 1



(d) Générateur 2 avec la perte sémantique 2



Chapitre 5

Synthèse des résultats

En résumé, la méthode exhaustive de génération des serpents et des forêts de serpents utilisant la fouille en profondeur est la plus simple d'implémentation et la plus intuitive. Par contre, il s'agit aussi de la méthode la moins efficace puisqu'elle ne permet aucune option d'élagage alors toutes les possibilités de serpents doivent être générées en entier afin de savoir si le serpent ou la forêt de serpents est maximal.

L'adaptation de la méthode des matrices de transfert [16][17] pour la génération des forêts de serpents maximales est un succès. Elle offre de nombreux avantages sur la méthode exhaustive utilisant la fouille en profondeur tels qu'un temps computationnel réduit et la possibilité d'être optimisée pour des structures ayant plus de contraintes comme des serpents maximaux ou des forêts où chaque ligne doit avoir un nombre minimum de cellules sélectionnées. Il est aussi possible de l'utiliser pour obtenir les séries génératrices des forêts de serpents pour les rectangles de base b si la matrice correspondante peut être inversée dans un délai raisonnable. De plus, les graphes résultant de cette méthode sont un outil supplémentaire pouvant être utilisé pour tenter d'autres architectures de réseaux neuronaux.

Pour ce qui est de l'utilisation de l'apprentissage machine, aucune des configurations de réseaux antagonistes génératifs n'a réussi à générer des serpents appartenant à des rectangles de dimensions supérieures à ceux utilisés lors de l'entraînement. De plus, aucune des stratégies (entrée conditionnelle, bruit structuré et perte sémantique) utilisées n'a permis de restreindre le réseau afin qu'il respecte la taille en pixels des cellules lors de la génération. Il semblerait donc que ce type de réseau neuronal soit incapable de résoudre un problème possédant un environnement variable. Toutefois, il est possible qu'une combinaison de l'utilisation d'images de plus grandes tailles ainsi que d'un ensemble d'entraînement contenant plus de dimensions et plus de serpents de chaque dimension permettent d'obtenir de meilleurs résultats. Par contre, en se basant sur les résultats de ce travail, il est difficile de croire que cela résoudrait le problème de génération des serpents pour des tailles de rectangles inconnues.

Aussi, la méthode de conversion des rectangles en images permet de remédier au problème de tailles de sortie variables. Elle pourrait donc être utilisée conjointement avec une différente architecture de réseau neuronal afin de générer des serpents dans des dimensions de rectangles variées et plus grandes que celles des données d'entraînement. Il est cependant important de se rappeler que cette stratégie peut produire une confusion dans le réseau puisqu'une image convertie en un rectangle de dimensions différentes que celles représentées peut parfois produire une serpent maximal valide pour cette mauvaise dimension.

Parmi les autres pistes n'ayant pas été explorées dans ce travail, il serait possible de tester avec des architectures de discriminateur plus complexes puisque cela n'a pas été effectué dans ce projet. Par exemple, il serait possible d'utiliser une combinaison de réseaux afin que l'un détermine si l'image est un serpent maximal et l'autre détermine si l'image représente le bon nombre de cellules. Cela pourrait peut-être remédier à la confusion causée par la conversion des images en rectangle et à l'incapacité du réseau à générer des images représentant des rectangles de plus grande dimension que les rectangles d'entraînement.

Chapitre 6

Conclusion

En conclusion, l'énumération des serpents et des forêts maximales est un problème combinatoire complexe toujours sans solution, mais non sans progrès.

Ce mémoire présente une version adaptée de la méthode des matrices de transfert qui permet de générer les graphes de génération des forêts de serpents. Dans le futur, il serait intéressant de tenter d'optimiser cette méthode afin de réduire le coût computationnel de la génération de ces graphes. Elle pourrait aussi être adaptée afin d'obtenir les graphes correspondant uniquement aux serpents. Il pourrait alors être intéressant de comparer les graphes des serpents à ceux des forêts afin de savoir s'ils seront réduits significativement. Notre hypothèse est que le nombre de noeuds risque de demeurer identique mais que le nombre d'arrête sera considérablement réduit, ce qui accélérerait significativement la recherche des serpents. De plus, la méthode des matrices de transfert permet d'introduire facilement des statistiques supplémentaires spécifiques comme le nombre de *kisses* (nombre de fois où deux cellules se touchent uniquement par le sommet).

Ce projet est aussi, à notre connaissance, un premier pas dans l'utilisation de

l'intelligence artificielle dans le domaine des polyominos, plus spécifiquement celui des serpents maximaux. Pour y faire suite, il serait bien d'explorer une approche d'apprentissage machine différente utilisant plutôt les graphes générés par la méthode des matrices de transfert. Une architecture intéressante est celle des *Hybrid Pointer Networks* [51], qui sont une amélioration des *Pointer Networks* ayant été publiée en 2021, soit au cours du projet actuel. Toutefois, il sera primordial de résoudre les problèmes soulevés dans la section 4.2 à propos de ce type de réseau et du contexte des serpents, soit l'obtention de réponses significatives et variées pour les hauteurs de rectangle désirées.

Chapitre 7

Bibliographie

[1] D. H. Redelmeier, "Counting polyominoes : Yet another attack," Discrete Mathematics, vol. 36, no. 2, pp. 191–203, 1981.

[2] I. Jensen, "Enumerations of lattice animals and trees," Journal of Statistical Physics, vol. 102, pp. 865–881, Feb 2001.

[3] I. Jensen, "Counting polyominoes : A parallel implementation for cluster computing," in Computational Science — ICCS 2003 (P. M. A. Sloot, D. Abramson, A. V. Bogdanov, Y. E. Gorbachev, J. J. Dongarra, and A. Y. Zomaya, eds.), (Berlin, Heidelberg), pp. 203–212, Springer Berlin Heidelberg, 2003.

[4] M. Bousquet-Mélou, "A method for the enumeration of various classes of column-convex polygons," Discrete Mathematics, vol. 154, no. 1, pp. 1–25, 1996.

[5] A. Goupil, M.-E. Pellerin, and H. Cloutier, "Generating functions for inscribed polyominoes," Discrete Applied Mathematics, vol. 161, no. 1-2, pp. 151–166, 2013. 151.

[6] OEIS Foundation Inc., "Number of recusive calls needed to compute the n-th Fibbonacci number F(n), starting with F(1) = F(2) = 1, Entry A019274 in The On-Line Encyclopedia of Integer Sequences." `http://oeis.org/A019274`, 2022.

[7] A. Goupil, M.-E. Pellerin, and J. W. de d'oplinter, "Partially directed snake polyominoes," Discrete Applied Mathematics, vol. 236, pp. 223–234, 2018.

[8] M. Bousquet-Mélou, "Families of prudent self-avoiding walks," Journal of Combinatorial Theory, Series A, vol. 117, no. 3, pp. 313–344, 2010.

[9] C. Barrientos and S. Minion, "The number of snakes in a box," 2018. 1.

[10] H. Lessard, "Programmes d'énumération de polyominos." UQTR, 2013.

[11] N. N. Madras and G. Slade, The self-avoiding walk. Probability and its applications, Boston : Birkhäuser, 1996.

[12] L. Danzer and V. Klee, "Lengths of snakes in boxes," Journal of Combinatorial Theory, vol. 2, no. 3, pp. 258–265, 1967.

[13] H. L. Abbott and M. Katchalski, "On the snake in the box problem," Journal of Combinatorial Theory, Series B, vol. 45, no. 1, pp. 13–24, 1988.

[14] D. Kinny, "A new approach to the snake-in-the-box problem," Frontiers in Artificial Intelligence and Applications, vol. 242, pp. 462–467, 01 2012.

[15] D. Zeilberger, "Symbol-crunching with the transfer-matrix method in order to count skinny physical creatures.," Integers, 2000.

[16] D. Zeilberger, "The umbral transfer-matrix method. i. foundations," Journal of Combinatorial Theory, Series A, vol. 91, no. 1, pp. 451–463, 2000.

[17] D. Zeilberger, "The umbral transfer-matrix method. iv. counting self-avoiding polygons and walks," Electronic Journal of Combinatorics, vol. 8, 09 2001.

[18] M.-E. Pellerin, "Énumération des polyominos d'index 0 à 2 à symétries près et des polycubes inscrits dans un prisme 2 x 2 x h." Mémoire de maîtrise, UQTR, 2015.

[19] Wikipedia contributors, "Depth-first search — Wikipedia, the free encyclopedia." `https://en.wikipedia.org/w/index.php?title=Depth-first_search&oldid=1067549728`, 2022. [Online ; accessed 3-February-2022].

[20] G. Kreweras, "Sur les partitions non croisees d'un cycle," Discrete Mathematics, vol. 1, no. 4, pp. 333–350, 1972.

[21] A. Goupil, "Document de travail." UQTR, 2022.

[22] OEIS Foundation Inc., "Maximum number of unit squares of a snake-like polyomino in an n x n square box, Entry A331968 in The On-Line Encyclopedia of Integer Sequences." `http://oeis.org/A331968`, 2022.

[23] K. Fukushima, "A neural network for visual pattern recognition," Computer, vol. 21, no. 3, 1988.

[24] S. Albawi, O. Bayat, S. Al-Azawi, and O. N. Ucan, "Social touch gesture recognition using convolutional neural network.," Computational Intelligence & Neuroscience, pp. 1–10, 2018.

[25] Y. S. Yang, Y. Liu, and P. M. Lam, "Self-avoiding walk model for proteins," Zeitschrift für Physik B Condensed Matter, vol. 59, pp. 445–447, Dec 1985.

[26] K. Huang, "Conditioned self-avoiding walk (csaw) : Stochastic approach to protein folding," Biophysical Reviews & Letters, vol. 2, no. 2, pp. 139–154, 2007.

[27] J. M. Bahi, C. Guyeux, K. Mazouzi, and L. Philippe, "Computational investigations of folded self-avoiding walks related to protein folding," Computational Biology and Chemistry, vol. 47, pp. 246–256, 2013.

[28] H. Chen, O. Engkvist, Y. Wang, M. Olivecrona, and T. Blaschke, "The rise of deep learning in drug discovery.," Drug discovery today, vol. 23, no. 6, pp. 1241–1250, 2018. 1241.

[29] S. D. Holcomb, W. K. Porter, S. V. Ault, G. Mao, and J. Wang, "Overview on deepmind and its alphago zero ai," in Proceedings of the 2018 International Conference on Big Data and Education, ICBDE '18, (New York, NY, USA), pp. 67–71, Association for Computing Machinery, 2018.

[30] S. Hochreiter and J. Schmidhuber, "Long short-term memory.," Neural Computation, vol. 9, no. 8, p. 1735, 1997.

[31] E. Azari and S. Vrudhula, "An energy-efficient reconfigurable lstm accelerator for natural language processing," in 2019 IEEE International Conference on Big Data (Big Data), pp. 4450–4459, 2019.

[32] W. Jin, R. Barzilay, and T. Jaakkola, "Junction tree variational autoencoder for molecular graph generation," in Proceedings of the 35th International Conference

on Machine Learning (J. Dy and A. Krause, eds.), vol. 80 of Proceedings of Machine Learning Research, pp. 2323–2332, PMLR, 10–15 Jul 2018.

[33] S. Purkayastha, I. Mondal, S. Sarkar, P. Goyal, and J. K. Pillai, "Drug-drug interactions prediction based on drug embedding and graph auto-encoder," in 2019 IEEE 19th International Conference on Bioinformatics and Bioengineering (BIBE), pp. 547–552, 2019.

[34] M. Zhang, S. Jiang, Z. Cui, R. Garnett, and Y. Chen, "D-vae : A variational autoencoder for directed acyclic graphs," in Advances in Neural Information Processing Systems (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[35] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," 2017.

[36] Q. Ma, S. Ge, D. He, D. Thaker, and I. Drori, "Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning," 2019.

[37] R. S. Sutton and A. G. Barto, Reinforcement learning : An introduction. Adaptive Computation and Machine Learning series, MIT Press, 2018.

[38] M. Wiering and M. van Otterlo, Reinforcement learning : state-of-the-art. Berlin ; : Springer, 2012.

[39] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play.," Science (New York, N.Y.), vol. 362, no. 6419, pp. 1140–1144, 2018. 1140.

[40] J. You, B. Liu, R. Ying, V. Pande, and J. Leskovec, "Graph convolutional policy network for goal-directed molecular graph generation," in Proceedings of the 32nd International Conference on Neural Information Processing Systems, NIPS'18, (Red Hook, NY, USA), pp. 6412–6422, Curran Associates Inc., 2018.

[41] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in Advances in Neural

Information Processing Systems (Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, eds.), vol. 27, Curran Associates, Inc., 2014.

[42] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," Communications of the ACM, vol. 63, no. 11, pp. 139–144, 2020. 139.

[43] S. Arora, R. Ge, Y. Liang, T. Ma, and Y. Zhang, "Generalization and equilibrium in generative adversarial nets (GANs)," in Proceedings of the 34th International Conference on Machine Learning (D. Precup and Y. W. Teh, eds.), vol. 70 of Proceedings of Machine Learning Research, pp. 224–232, PMLR, 06–11 Aug 2017.

[44] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in Proceedings of the 34th International Conference on Machine Learning - Volume 70, ICML'17, pp. 214–223, JMLR.org, 2017.

[45] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks : An overview," IEEE Signal Processing Magazine, vol. 35, no. 1, 2018.

[46] M. Mirza and S. Osindero, "Conditional generative adversarial nets," 2014.

[47] L. Di Liello, P. Ardino, J. Gobbi, P. Morettin, S. Teso, and A. Passerini, "Efficient generation of structured objects with constrained adversarial networks," in Advances in Neural Information Processing Systems (H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds.), vol. 33, pp. 14663–14674, Curran Associates, Inc., 2020.

[48] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in ICML Workshop on Deep Learning for Audio, Speech and Language Processing, 2013.

[49] G. E. Hinton, S. Nitish, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," 2012.

[50] J. Xu, Z. Zhang, T. Friedman, Y. Liang, and G. Van den Broeck, "A semantic loss function for deep learning with symbolic knowledge," in Proceedings of the

35th International Conference on Machine Learning (J. Dy and A. Krause, eds.), vol. 80 of Proceedings of Machine Learning Research, pp. 5502–5511, PMLR, 10–15 Jul 2018.

[51] A. Stohy, H.-T. Abdelhakam, S. Ali, M. Elhenawy, A. A. Hassan, M. Masoud, S. Glaser, and A. Rakotonirainy, "Hybrid pointer networks for traveling salesman problems optimization.," PloS one, vol. 16, no. 12, p. e0260995, 2021. 0260995.

Appendices

Annexe A

Algorithmes

A.1 Algorithme d'énumération des serpents maximaux ou des forêts maximales cellule par cellule

L'algorithme 2 permet de rechercher exhaustivement tous les serpents maximaux ou toutes les forêts de serpents maximales d'une grille. Les changements sont précisés dans la sous-section 3.2.1 pour les serpents maximaux et dans la sous-section 3.2.2 pour les forêts de serpents maximales.

Définitions :

— *grille* : Grille de recherche correspondant au rectangle $b \times h$. Une cellule aura la valeur 1 si elle est sélectionnée et 0 si elle ne l'est pas ;

— *maximaux* : Liste des grilles contenant des serpents maximaux. Pour alléger le pseudocode, elle aura la propriété *max* qui sera la longueur des serpents (ou forêts) maximaux ;

— *maximauxLocaux* : Liste des grilles contenant les serpents maximaux à partir d'un parcours partiel du graphe. Possède aussi la propriété *max*.

Algorithm 2 Énumeration des serpents maximaux

Require: *grille*

```
1: procedure RECHERCHE
2:     maximaux ← empty list
3:     for all cell_ij in grille do
4:         cell_ij ← 1
5:         RECHERCHEREC(grille, maximaux, i, j)
6:         cell_ij ← 0
7:     return maximaux
8: procedure RECHERCHEREC(grille, maximaux, i, j)
9:     for all cell_kl in cells_to_search_from_ij do          ▷ Voir figure 3.1
10:        if eligible then                                   ▷ Selon les critères spécifiques
11:            cell_kl ← 1
12:            maximauxLocaux ← RECHERCHEREC(grille, maximaux, k, l)
13:            if maximaux.max > maximauxLocaux.max then
14:                maximaux ← maximauxLocaux
15:            else if maximaux.max = maximauxLocaux.max then
16:                maximaux.ajouter(maximauxLocaux)
17:            cell_kl ← 0
```

A.2 Algorithme d'adjacence entre un noeud d'entrée et de sortie

Définitions :

— *entree* : Le noeud d'entrée ;

— *sortie* : Le noeud de sortie ;

— i : L'indice de colonne ;

— a : La valeur a est le nombre de cellules sélectionnées ;

— c : La valeur c est le nombre de composantes connexes ;

— *comp* : Valeur booléenne étant vraie si la composante connexe parcourue est nouvelle ;

— cycle : Valeur booléenne étant vraie si la combinaison des noeuds crée un cycle. La présence de cycle est déterminée tel que décrit dans la sous-section 3.3.2.

De plus, la vérification mentionnée dans les deux fonctions est celle présentée dans la partie **Première fonction** de la sous-section 3.3.3, soit :

— L'interdiction d'ajouter une cellule sélectionnée sur une cellule de degré 2 ;

— L'interdiction d'ajouter une cellule ayant une étiquette de liaison sur une cellule vide, sauf si la cellule ajoutée est une cellule isolée sur la ligne ;

— L'interdiction d'ajouter une cellule sans étiquette de liaison sur une cellule sélectionnée ;

— La validation de l'étiquette de liaison, si la cellule ajoutée en possède une.

Algorithm 3 Adjacence entre un noeud d'entrée et de sortie

Require: *entree*, *sortie*

```
1: procedure FONCTION1(entree, sortie)
2:     i ← 0
3:     a ← 0
4:     c ← 0
5:     if cycle then
6:         return null, null
7:     while i < longueur_du_noeud do
8:         cellule ajoutée = sortie[i]
9:         cellule d'entrée = entree[i]
10:        comp ← false
11:        if vérification échouée then
12:            return null, null
13:        if cellule ajoutée sélectionnée then
14:            a ← a + 1
15:            if cellule d'entrée non sélectionnée then
16:                comp ← true
17:            if i + 1 < longueur_du_noeud then
18:                i, a, c ← FONCTION2(entree, sortie, i + 1, a, c, comp)
19:                if i est null then
20:                    return null, null
21:                i ← i − 1
22:            else if comp then
23:                c ← c + 1
24:    return a, c
25: procedure FONCTION2(entree, sortie, i, a, c, comp)
26:    cellule ajoutée = sortie[i]
27:    cellule d'entrée = entree[i]
28:    if vérification ok then
29:        return null, null, null
30:    if comp then
31:        if cellule ajoutée non sélectionnée then
32:            c ← c + 1
33:        else if cellule d'entrée sélectionnée then
34:            comp ← false
35:    else if cellule ajoutée sélectionnée and cellule d'entrée sélectionnée then
36:        c ← c − 1
37:    if cellule ajoutée non selectionnée then
38:        return i + 1, a, c
39:    else if i + 1 < longueur_du_noeud then
40:        return FONCTION2(entree, sortie, i + 1, a, c, comp)
41:    else if comp and cellule d'entrée non sélectionnée then
42:        c ← c + 1
43:    return i + 1, a, c
```

Annexe B

Nombre de serpents maximaux et de forêts maximales libres et fixes

TABLE B.1 – Nombre de serpents maximaux libres et fixes par dimensions de rectangles

	2xh		**3xh**		**4xh**		**5xh**		**6xh**		**7xh**	
h	**Libres**	**Fixes**	**Libres**	**Fixes**	**Libres**	**Fixes**	**Libres**	**Fixes**	**Libres**	**Fixes**	**Libres**	**Fixes**
2	1	4										
3	1	2	3	8								
4	2	6	4	14	21	84						
5	1	2	6	18	7	26	16	56				
6	2	8	1	2	8	32	1	4	34	136		
7	1	2	1	4	5	16	6	24	3	10	16	52
8	3	10	5	6	38	152	8	32	42	168	2	4
9	1	2	2	8	12	48	27	108	9	32	2	8
10	3	12	3	10	6	24	1	2	2	8	55	200
11	1	2	3	12	84	336	2	6	7	20	213	852
12	4	14	4	14	16	64	2	6	111	444	8	32
13	1	2	4	16	9	32	14	48	18	72	32	124
14	4	16	5	18	142	568	38	138	6	24		
15	1	2	5	20	20	80	73	274	9	30		
16	5	18	6	22	10	40	134	524	235	940		
17	1	2	6	24	212	848	431	1696	47	184		
18	5	20	7	26	24	96	994	3930	8	32		
19	1	2	7	28	13	48	1837	7268	16	50		
20	6	22	8	30	294	1176	12	48				
21	1	2	8	32	28	112						
22	6	24	9	34	14	56						
23	1	2	9	36	388	1552						
24	7	26	10	38	32	128						
25	1	2	10	40	17	64						
26	7	28	11	42	494	1976						
27	1	2	11	44	36	144						
28	8	30	12	46	18	72						
29	1	2	12	48	612	2448						
30	8	32	13	50	40	160						
31	1	2	13	52	21	80						
32	9	34	14	54	742	2968						
33	1	2	14	56	44	176						
34	9	36	15	58	22	88						
35	1	2	15	60	884	3536						

TABLE B.2 – Nombre de forêts de serpents maximales libres et fixes par dimensions de rectangles

	2xh		**3xh**		**4xh**		**5xh**		**6xh**		**7xh**		**8xh**	
h	**Libres**	**Fixes**	**Libres**	**Fixes**	**Libres**	**Fixes**	**Libres**	**Fixes**	**Libres**	**Fixes**	**Libres**	**Fixes**	**Libres**	**Fixes**
2	1	4												
3	1	2	3	8										
4	3	8	4	14	32	128								
5	1	2	9	27	23	79	98	358						
6	3	12	1	2	13	52	8	30	26	104				
7	1	2	1	4	5	16	60	217	27	104	537	2092		
8	5	16	2	6	119	456	1	2	12	48	55	204	166	664
9	1	2	4	12	43	170	8	30	1	2	15	58	7	28
10	5	20	7	22	12	42	83	314	37	148	3	8	52	196
11	1	2	11	36	467	1804	1	2	15	60	70	266	1	2
12	7	24	19	62	113	452	10	40	1	2	18	72	8	32
13	1	2	32	112	27	92	106	400	55	220	3	8	60	224
14	7	28	55	198	1446	5714	1	2	22	88	78	290	1	2
15	1	2	95	348	284	1130	12	44	1	2	18	72	9	36
16	9	32	165	622	52	192	132	504	67	268	3	8	62	232
17	1	2	292	1112	4107	16240	1	2	27	104	79	298	1	2
18	9	36	511	1974	682	2728	12	48	1	2	18	72	9	36
19	1	2	901	3508	107	392	160	608	82	322	3	8	63	234
20	11	40	1592	6246	10869	43290	1	2	30	120	82	306	1	2
21	1	2	2818	11104	1608	6416	14	52	1	2	18	72	9	36
22	11	44	4984	19726	207	792	187	720	94	376	3	8	63	234
23	1	2	8841	35068	27768	110596	1	2	35	136	83	314	1	2
24	13	48	15681	62350	3696	14784	14	56	1	2	18	72	9	36
25	1	2	27836	110824	417	1592	219	840	109	430	3	8	63	234
26	13	52		196998		274072	1	2	38	152	86	322	1	2
27	1	2		350228	8387	33512	16	60	1	2	18	72	9	36
28	15	56		622614	817	3192	250	968	121	484	3	8	63	234
29	1	2		1106800		663696	1	2	43	168	87	330	1	2
30	15	60		1967582	18740	74960	16	64	1	2	18	72	9	36
31	1	2		3497836	1637	6392	286	1104	136	538	3	8	63	234
32	17	64		6218110		1578224	1	2	46	184	90	338	1	2
33	1	2			41481	165848	18	68	1	2	18	72	9	36
34	17	68			3237	12792	321	1248	148	592	3	8	63	234
35	1	2				3697768	1	2	51	200	91	346	1	2

Annexe C

Architectures du GAN

Cette annexe présente les différentes architectures des générateurs testés ainsi que celle du discriminateur, qui a été la même pour tous les tests. Tous les modèles ont été réalisés avec Python 3.8 et Tensorflow 2.6.0.

FIGURE C.1 – Schéma des générateurs où i et j sont les dimensions des images

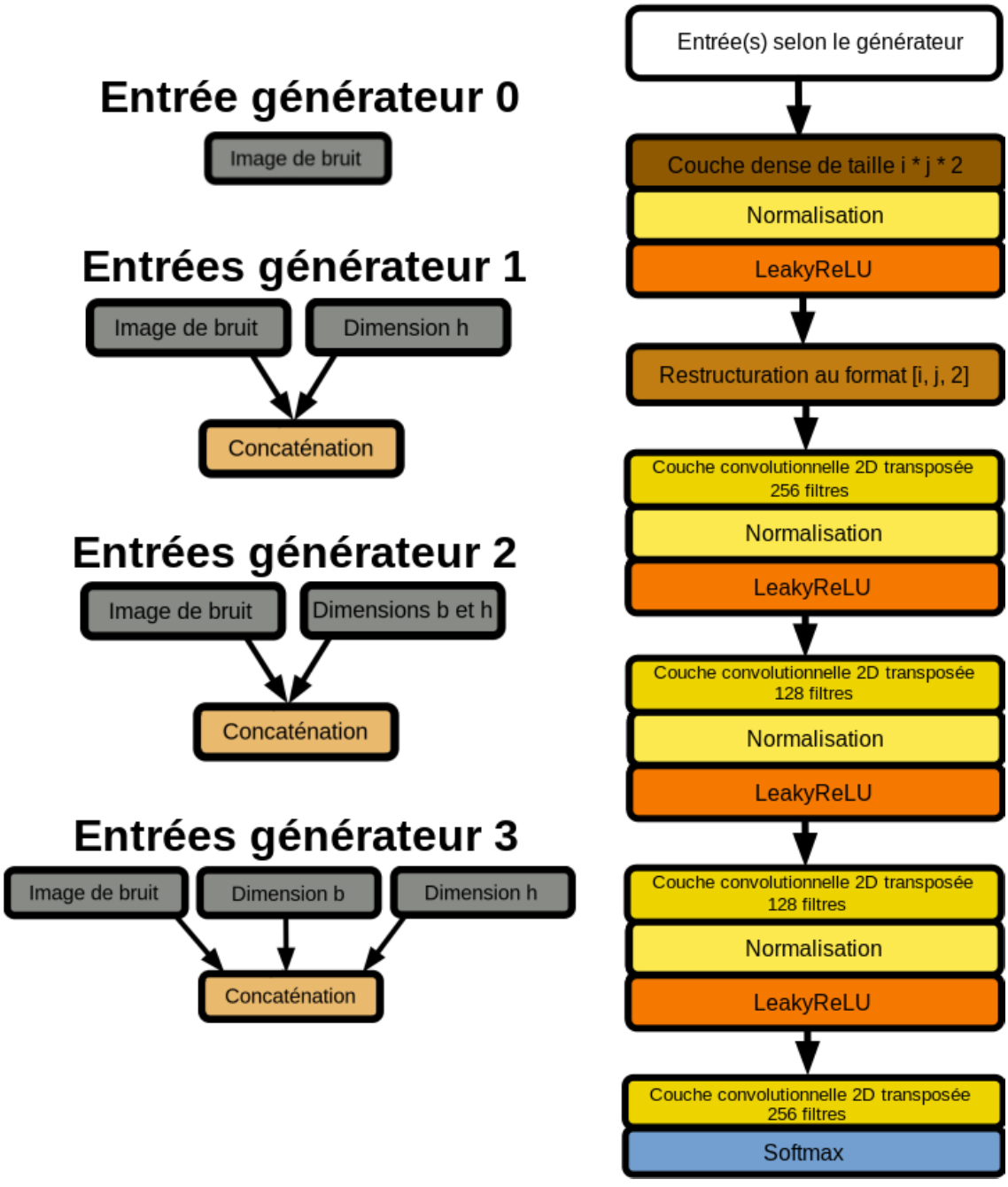
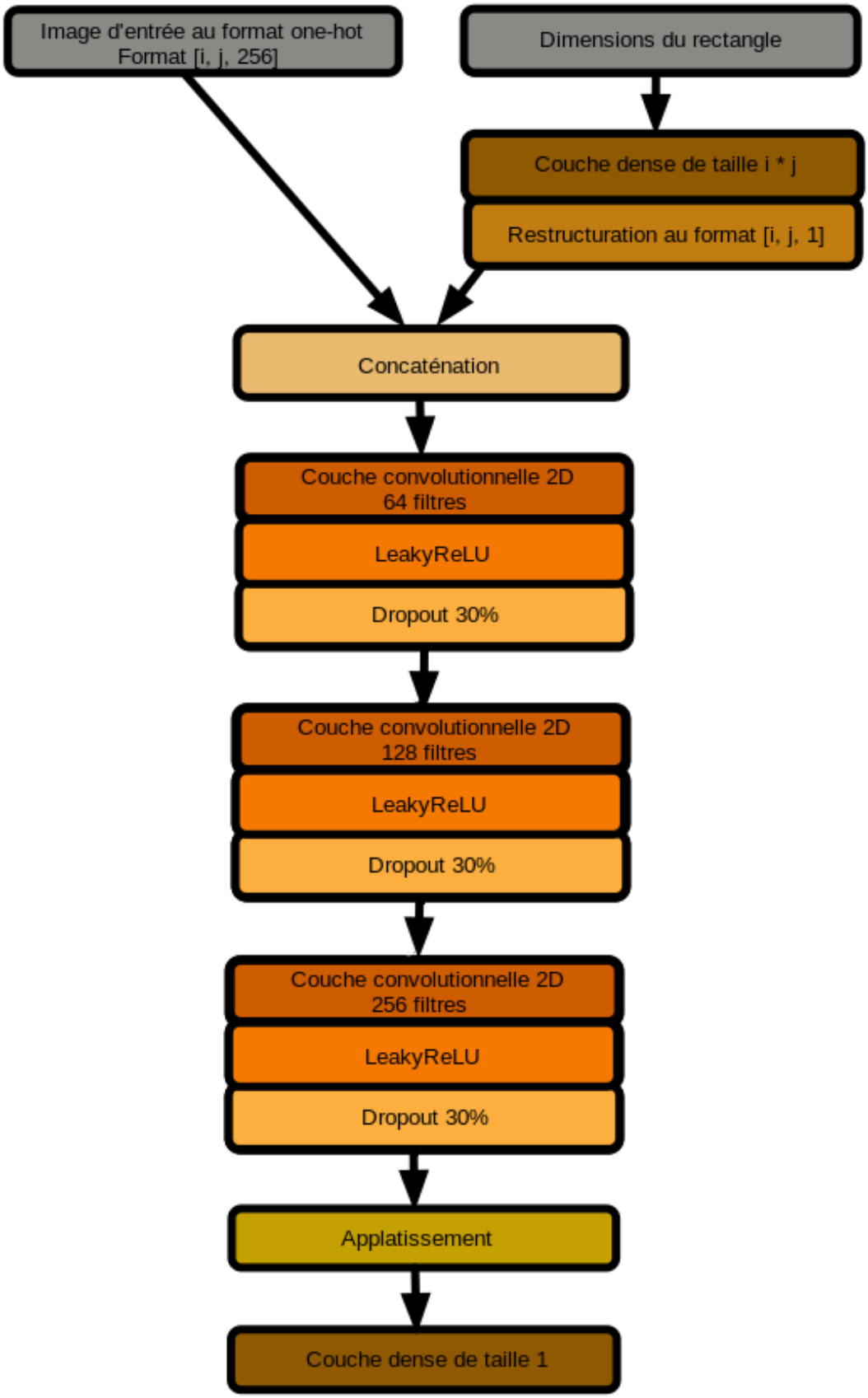


FIGURE C.2 – Schéma du discriminateur où i et j sont les dimensions des images

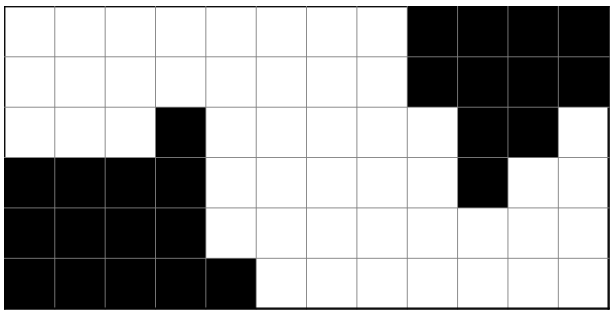


Annexe D

Exemple d'estimation des dimensions d'un rectangle dans une image pour le calcul de SEM1

Prenons comme exemple l'estimation de la hauteur du rectangle représenté dans l'image de la figure D.1.

FIGURE D.1 – Image représentant un rectangle aux dimensions à estimer



La première étape est de calculer le nombre d'occurrences du nombre de cellules consécutives qui sont dans le même état. Le résultat de ce décompte est présenté dans le tableau D.1.

TABLE D.1 – Nombre d'occurences de chaque nombre de pixels consécutifs ayant le même état par colonne de pixels de l'image de la figure D.1

Nombre de pixels	Nombre d'occurrences par colonne	Total d'occurrences
1	1	1
2	1 1 1	4
3	2 2 2 2	8
4	1 1 1	4
5	1	1
6	1 1 1	3

Ensuite, on permet une marge d'erreur de plus ou mois un pixel dans la séparation entre les cellule. Pour cela, on additionne les occurences des nombres de pixels ayant plus ou moins un pixel de différence afin de déterminer la nombre de pixels de hauteur le plus probable. de cellule la plus probable. Ainsi, par exemple, le nombre ajusté d'occurrences de 3 pixels consécutifs sera la somme du nombre d'occurrences de 2, 3 et 4 pixels consécutifs. Ces résultats pour l'exemple actuel sont présenté dans le tableau D.2.

TABLE D.2 – Total du nombre d'occurrences réel et du nombre d'occurrences incluant la marge d'erreur pour l'image de rectangle de la figure D.1

Hauteur de cellule	Total d'occurrences	Total d'occurrences incluant une marge d'erreur de ± 1 pixel
1	1	5
2	4	13
3	8	16
4	4	13
5	1	8
6	3	4

Pour continuer, on calcule la hauteur moyenne en pixels des cellules en effectuant une moyenne pondérée des occurrences réelles de la hauteur la plus probable, soit celle ayant le plus grand total d'occurrences incluant la marge d'erreur, et des hauteurs plus ou moins un. Cela est fait selon l'équation D.1 où h_p est la hauteur la plus probable et n est le nombre d'occurrences réelles d'une hauteur.

$$
\begin{aligned}
h_{cellule} &= \frac{(h_p - 1) * n_{h_p-1} + h_p * n_{h_p} + (h_p + 1) * n_{h_p+1}}{n_{h_p-1} + n_{h_p} + n_{h_p+1}} \\
&= \frac{2 * 4 + 3 * 8 + 4 * 4}{4 + 8 + 4} \\
&= 3
\end{aligned}
\tag{D.1}
$$

Finalement, on obtient la hauteur du rectangle après avoir arrondi à l'unité près le résultat de la division de la hauteur de l'image par la hauteur de cellule moyenne obtenue précédemment. Donc pour cet exemple, la hauteur estimée du rectangle est de deux cellules puisque $h = 6/3 = 2$.

Annexe E

Tableau de validation par dimension de rectangle

Le tableau E.1 présente les pourcentages de validation obtenus par les différents modèles pour chacune des dimensions de rectangle allant de 4×4 à 4×16. Il est possible d'y observer que les dimensions 4×7, 4×12 et 4×16 sont particulièrement problématique pour la majorité des modèles.

Afin de réduire la taille des tableaux et des graphiques, les modèles sont identifiés par trois chiffres représentant respectivement le numéro de configuration, le numéro du générateur et le numéro de la perte sémantique (0 est utilisé lorsqu'aucune perte sémantique est appliquée). Ainsi, le modèle 2-0-1 est le modèle utilisant la configuration 2, le générateur 0 et la SEM1. Le chiffre correspondant à la configuration peut être omis lorsqu'elle est précisée explicitement.

TABLE E.1 – Pourcentages de validation par dimension de rectangles après 2000 époques

	4x4	**4x5**	**4x6**	**4x7**	**4x8**	**4x9**	**4x10**	**4x11**	**4x12**	**4x13**	**4x14**	**4x15**	**4x16**
0-0-0	0	0	0	0	0	0	1,37	7,5	1,41	32,58	32,05	0	0
0-0-1	0	0	0	0	0	0	0	1,25	1,09	37,5	42,5	0	0
0-0-2	0	0	0	0	0	0	0	4,55	5,41	18,33	21,59	0	0
0-1-0	78,16	55,29	44,29	0	36,25	20	17,57	56,63	22,03	20,51	75,95	39,44	0
0-1-1	0	0	29,63	19,1	62,86	11,25	17,65	61,25	1,64	9,84	79,76	52,87	0
0-1-2	0	0	59,72	2,56	61,18	16,84	3,7	45,33	22,73	0	74,65	49,38	0
0-2-0	87,3	74,07	43,84	0	36	16,44	6,76	50	28,09	1,15	76,92	31,03	0
0-2-1	0	0	0	48,78	77,78	22,39	9,09	53,16	27,27	2,78	83,15	42,68	0
0-2-2	0	0	62,96	0	68,35	26,67	23,53	66,67	11,94	40,58	80,26	46,34	0
0-3-0	0	0	0	27,27	53,52	21,21	22,22	47,06	6,98	7,14	54,69	17,46	1,1
0-3-1	82,19	0	1,09	0	0	47,56	9,33	47,3	14,67	6,33	77,38	41,46	3,51
0-3-2	81,72	60,94	15,12	0	37,21	33,33	20,83	46,58	12,36	18,89	60,29	39,73	0
2-0-0	52,33	22,97	2,78	0	22,73	17,5	30,43	45,12	1,27	9,68	47,54	3,7	0
2-0-1	19,77	54,05	0	0	15,15	15	18,84	32,93	1,27	6,45	39,34	2,47	0
2-0-2	23,26	12,16	0	0	22,73	13,75	11,59	48,78	5,06	1,08	57,38	11,11	0
2-1-0	32,56	25,68	4,17	0	27,27	30	27,54	30,49	0	11,83	57,38	0	0
2-1-1	40,7	25,68	0	0	24,24	16,25	23,19	32,93	2,53	10,75	62,3	4,94	0
2-2-0	39,53	13,51	1,39	0	33,33	23,75	18,84	32,93	2,53	6,45	65,57	1,23	0
4-2-1	0	0	0	1,11	73,68	21,95	20	52,7	8	24,05	79,76	43,9	0
4-2-2	0	0	68,48	0	28,95	7,32	42,67	39,19	0	18,99	46,43	14,63	0
5-0-1	88,37	5,41	0	0	0	26,25	20,29	53,66	2,53	11,83	52,46	2,47	0
5-0-2	33,72	16,22	2,78	0	12,12	18,75	20,29	29,27	2,53	13,98	45,9	0	0
6-0-1	36,05	17,57	1,39	0	7,58	32,5	8,7	30,49	7,59	8,6	60,66	1,23	0
6-0-2	19,77	8,11	1,39	0	12,12	26,25	8,7	30,49	5,06	2,15	62,3	2,47	0
7-2-1	97,26	80,33	52,17	0	21,05	29,27	9,33	51,35	8	2,53	86,9	43,9	0
7-2-2	0	90,16	1,09	0	55,26	37,8	61,33	72,97	5,33	18,99	80,95	47,56	0
8-2-1	0	0	0	60	64,47	14,63	17,33	52,7	4	30,38	83,33	53,66	0
8-2-2	0	77,05	0	0	36,84	26,83	29,33	56,76	14,67	21,52	82,14	45,12	0
9-0-1	20,93	9,46	1,39	0	7,58	28,75	21,74	34,15	0	8,6	40,98	1,23	0
9-0-2	47,67	16,22	1,39	0	4,55	32,5	33,33	37,8	1,27	9,68	63,93	1,23	0
10-0-0	90,24	86,96	96,63	82,54	77,89	83,95	82,14	54,55	9,38	83,54	62,5	64,38	46,38
11-0-0	0	0	0	0	0	0	1,33	62,16	0	0	0	0	0
11-1-0	98,63	98,36	94,57	0	73,68	65,85	60	91,89	16	92,41	73,81	96,34	0
11-2-0	86,3	95,08	95,65	0	53,95	64,63	84	83,78	2,67	96,2	76,19	97,56	0
11-3-0	100	95,08	98,91	15,56	86,84	63,41	92	90,54	0	97,47	86,9	96,34	0
12-2-0	73,97	0	93,48	0	76,32	80,49	38,67	72,97	90,67	0	0	91,46	0