Échantillonnage pour les jeux vidéo

Julien Gheysens

Université de Lille

Laboratoire Paul Painlevé

Université de Lille

Thèse

pour obtenir le grade de :

Docteur de l'Université de Lille

dans la spécialité

« Mathématiques et leurs interactions »

Probabilité et Statistiques

par

Julien GHEYSENS

Échantillonnage pour les jeux vidéo

Thèse soutenue en 21 décembre 2023 devant le jury composé de :

M.	Jérôme Lelong	Professeur, Labo Jean Kuntzmann, Université de Grenoble INP	(Rapporteur)
Mme	Peggy Cénac	Maîtresse de conférences, HDR, Institut Mathématiques de Bourgogne, Université de Bourgogne	(Rapporteuse)
Mme	Charlotte Baey	Maîtresse de conférences, Labo Paul Painlevé, Université de Lille	(Examinatrice)
M.	Serguei Dachian	Professeur, Labo Paul Painlevé, Université de Lille	(Examinateur et Président du jury)
Mme	Cécile Mailler	Maîtresse de conférences, Probability Lab at Bath, Université de Bath	(Examinatrice)
M.	David Coupier	Professeur, IMT Nord Europe, Université de Lille	(Co-Directeur de Thèse)
M.	Nicolas Wicker	Professeur, Labo Paul Painlevé, Université de Lille	(Directeur de Thèse)

Laboratoire Paul Painlevé
École doctorale MADIS

Résumé

Ces dernières années, les jeux vidéo ont pris de plus en plus d'ampleur dans le domaine du divertissement. Une multitude de programmes ludiques sont diffusés chaque jour sur les différentes plateformes de téléchargement. D'où l'importance pour les développeurs d'essayer de se démarquer. Dans cette thèse, nous avons utilisé la méthode de Metropolis-Hasting pour réaliser des échantillons de polyominos ou de sous-ensembles de personnages, deux éléments particulièrement bien adaptés pour ce domaine. Une étude sur la convergence de ces algorithmes a aussi été réalisée pour aider la compréhension. De plus, une étude sur un estimateur des valeurs de Shapley est aussi proposée pour avoir un outil critique sur l'intelligence artificielle.

Abstract

In recent years, video games have become more and more important in the entertainment field. Many fun programs are broadcast every day on the different download platforms. Hence developers must try to stand out. In this thesis, we have used the Metropolis-Hasting method to produce samples of polyominoes or subsets of characters, two elements particularly well suited in this field. A study on the convergence of these algorithms has also been carried out to help understanding. In addition, a study on an estimator of Shapley values is also proposed to have a critical tool for artificial intelligence.

Remerciements

Tout d'abord, je souhaite remercier mes directeurs de thèse David COUPIER et Nicolas WICKER qui m'ont accompagnés ces cinq dernières années tout au long de mon doctorat et des différents projets. Je remercie chaleureusement Peggy CÉNAC et Jérôme LELONG pour avoir accepté le statut de rapporteuse et rapporteur pour cette thèse, et de leur temps qu'il me consacre. Je remercie également Charlotte BAEY, Serguei DACHIAN et Cécile MAILLER de participer à mon jury.

Je tiens à remercier l'équipe de Proba-Stats du laboratoire Paul Painlevé pour leur accueil et leur bienveillance à mon égard. Je remercie aussi l'équipe d'Xperium qui m'a donnée l'occasion de m'entrainer en tant qu'orateur pour de la vulgarisation scientifique. Je profite aussi pour remercier Joël DUMONT, enseignant du secondaire, pour m'avoir sensibilisé et donné envie de continuer d'étudier les mathématiques.

Ensuite, je remercie mes parents Caroline et Dominique GHEYSENS et mon frère Tanguy GHEYSENS qui m'ont toujours soutenus dans mes projets. Je remercie aussi ma compagne Blandine DIMPRE qui est présente pour moi au quotidien.

Enfin, je dédie cette thèse à mon tout jeune fils Grégoire.

Table des matières

Table des figures

Introduction

Ces dernières années, les jeux vidéo ont pris de plus en plus d'ampleur dans le domaine du divertissement. Une multitude de programmes ludiques sont diffusés chaque jour sur les différentes plateformes de téléchargement. Une catégorie qui nous paraît intéressante est la catégorie *"RogueLike"*. Elle regroupe un ensemble de jeux vidéo dans lequel le joueur doit parcourir un donjon rempli de défis ou d'ennemis. Généralement, le donjon peut être totalement exploré rapidement. Cependant, ces jeux disposent d'une rejouabilité plus importante car les donjons sont générés procéduralement. À chaque nouvelle partie, les joueurs peuvent explorer un donjon différent par rapport à leurs parties précédentes. Le nom de cette catégorie est tiré du jeu *Rogue* (1980) qui a popularisé la catégorie. Aujourd'hui, on compte près de 3.000 jeux du type *"RogueLike"* sur la plateforme *Steam*. D'où l'importance pour les développeurs d'essayer de se démarquer. Une possibilité de démarcation est d'enrichir l'aspect environnement aléatoire aux travers d'outils que nous allons expliquer dans ce document.

Dans cette thèse, nous allons utiliser la méthode de Metropolis-Hasting pour réaliser deux algorithmes d'échantillonnage. Dans la première partie, nous nous concentrons sur les polyominos. Avec un développement sur la notion d'*épluchage* d'un polyomino, nous allons mettre en place un algorithme qui converge vers la loi uniforme sur l'ensemble des polyominos de taille donnée. Il sera ensuite utilisé dans un contexte de simulation de la ville de Lille du XVIIIème siècle. Ensuite, dans notre deuxième partie, nous allons développer un nouvel algorithme qui nous permettra de sélectionner aléatoirement des équipes de héros pour le jeu vidéo *Dota 2*. Cette fois-ci, l'algorithme convergera vers une loi cible extraite d'un apprentissage sur les scores des parties d'une base de données publique. Enfin, notre dernière partie se concentrera sur l'élaboration d'un estimateur basé sur un processus ponctuel déterminantal pour les valeurs de Shapley. Elles nous permettront de mieux comprendre nos futurs systèmes d'apprentissage qui ont été entrainés dans des environnements enrichis grâce aux méthodes d'échantillonnage des deux

premières parties. Une critique de notre estimateur sera menée pour observer ses performances dans deux contextes : sur une base de données du jeu *League of Legends* et une autre sur des contenus *Instagram*. Mais avant de présenter les différents résultats obtenus, nous allons faire un rapide résumé des différentes thèmes et projets abordés durant ces trois années.

Projets de jeu vidéo

ChickenPods est un jeu vidéo sur mobile où le but est de développer une population de poules qui évoluent dans un environnement généré aléatoirement. En l'occurrence, la carte du jeu est construite à chaque nouvelle partie et repose sur deux algorithmes principaux. Un processus ponctuel déterminantal positionne les zones de nourriture et plusieurs marches aléatoires auto-évitantes avec plusieurs copules définissent le labyrinthe de barrières dans le jeu. Outre les animations de vulgarisation des mathématiques dans les jeux vidéo pour un jeune public, un encadrement du développement du jeu avec *Unreal Engine* a pu être réalisé.

Mais ce n'est pas le seul projet de jeu vidéo. Liées aux recherches sur les polyominos du premier chapitre de cette thèse, un autre projet a commencé lors de ces trois dernières années. *Lille18* est un jeu de rôle dans la ville de Lille à la fin du XVIII^{e} siècle. L'objectif derrière ce projet est de tester des algorithmes de génération aléatoires et d'apprentissage automatiques avec le standard des jeux vidéo modernes. Une première version de ce jeu a pu être faite sous *Unreal Engine*. Elle se concentre sur un unique quartier de cette ville. Cette version nous a notamment permis de tester des algorithmes d'intelligence artificielle pour des personnages combattant aux corps à corps. A partir d'un environnement simplifié en python, nous avons pu entrainer un réseau de neurones que nous avons transféré ensuite dans le moteur de jeu.

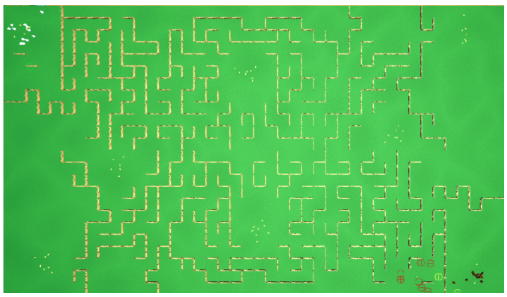

FIGURE 1 – Exemple d'environnement généré dans *ChickenPods*

FIGURE 2 – Capture du jeu *Lille18*(en cours de développement)

Apprentissage par renforcement supervisé

Un autre thème abordé durant cette thèse est celui de l'apprentissage automatique. Parmi les différentes catégories existantes, deux d'entre elles ont retenu notre attention : l'apprentissage supervisé et l'apprentissage par renforcement.

L'apprentissage supervisé consiste à entrainer un agent à partir de données qui vont servir d'exemples. Etant donné un environnent de jeu, l'agent évolue et apprend en fonction des données qu'on lui fournit ou par mimétisme d'un autre agent que l'on souhaite approcher. Quant à lui, l'apprentissage par renforcement repose sur le fait d'entrainer un agent à partir d'un système de récompense. Pour cela, nous avons besoin d'un environnement détaillé qui va pouvoir simuler les changements d'état d'un agent en fonction des décisions qu'il prend.

Notre idée ici est d'associer les deux méthodes dans le contexte des jeux vidéo. Ayant un environnement détaillé et une solution pouvant être mise en place relativement rapidement, les contraintes des deux catégories d'apprentissage cités peuvent être satisfaites. En effet, en s'appuyant sur les algorithmes d'intelligence artificielle implémentés dans les moteurs des jeux vidéo, nous pouvons avoir un agent qui va nous fournir des exemples d'actions souhaités. Mais pour compléter l'apprentissage, nous pouvons utiliser la méthode par renforcement. A partir de l'environnement de jeu du moteur, nous pouvons lui associer un système de récompense. Ainsi, en utilisant cette méthode, nous pouvons avoir potentiellement des agents robustes qui peuvent faire des choix standard et qui sont adaptés à l'environnement.

Pour nos premiers tests empiriques, nous nous sommes basés sur l'algorithme *Actor-Critic* [13] qui dispose de deux réseaux de neurones. Le premier permet de décider quelle action doit être choisi pour notre agent (l'*actor*), et le second va faire une évaluation de l'action prise (le *critic*). Pour l'environnement, nous avons sélectionné celui de *LunarLanderContinuous* présent dans la librairie *Gym*. Le dernier élément nécessaire est l'agent qui va nous servir de modèle pour la phase supervisée. Pour simplifier le problème, nous avons utilisé dans un premier temps un agent, appelé agent maître, qui a appris avec l'algorithme *Actor-critic*.

Avec les éléments expliqués précédemment, l'apprentissage par renforcement supervisé ne nous donne pas les résultats attendus. En effet, bien que la phase supervisée permet d'augmenter le gain moyen de notre agent, lorsqu'on débute la phase de renforcement, le gain moyen commence à diminuer. Pour retrouver un gain équivalent au gain moyen de l'agent maître, notre agent a besoin d'un temps relativement similaire à celui nécessaire pour l'apprentissage de l'agent maître. Par conséquent, la phase supervisée semble ne pas apporter de plus-value. Un complément de recherche sur le sujet est envisagé après la thèse.

Chapitre 1

Échantillonnage de Polyominos

Dans ce chapitre, nous allons nous concentrer sur le développement d'un algorithme nous permettant d'échantillonner aléatoirement des polyominos de taille fixée de façon uniforme. Cet objectif résulte d'une recherche de génération aléatoirement de cycles dans un maillage donné, en l'occurrence pour le plan de la ville au XVIIIème siècle. Après plusieurs tentatives, la manipulation des cycles s'est avérée complexe. Dans une démarche de simplification du problème, nous nous sommes concentrés sur la génération de polyominos. L'idée étant qu'à partir d'un ensemble de polyominos, il nous est possible de modifier cet ensemble en un autre ensemble mais cette fois-ci de cycles. Une manière d'y arriver est de considérer seulement pour un certain polyomino son périmètre extérieur. Nous expliciterons cette méthode dans à la fin de ce chapitre. Bien que l'ensemble de cycles aléatoires ainsi produit n'est pas un sous-ensemble uniforme sur l'ensemble des cycles possibles, il nous permet tout de même d'accéder à une génération uniforme d'un sous-ensemble des cycles possibles.

Après une introduction des éléments de notre problème, nous allons détailler les méthodes existantes qui vont nous aider à construire notre algorithme d'échantillonnage. Ensuite, une étude sur les caractéristiques des polyominos sera faite pour donner des informations sur la convergence de notre algorithme vers la loi cible, la loi uniforme sur l'ensemble des polyominos de même taille. Enfin, nous utiliserons ce que nous auront démontré dans le contexte d'une simulation de la ville de Lille. C'est à cette instant que nous préciserons comment il nous est possible de transformer un ensemble aléatoire de polyominos en un autre ensemble de cycles.

1.1 Problème d'échantillonnage

1.1.1 Les chaînes de Markov

Soit Ω un ensemble fini d'éléments (potentiellement très grand) et π une distribution de probabilité sur cet ensemble. De manière usuelle, le problème d'échantillonnage réside dans la façon dont on peut sélectionner un élément de Ω aléatoirement selon la distribution π. Ce type de problème peut avoir des applications dans la recherche de *comptage approximatif* ou en *physique statistique*. Pour la résolution de ce problème, la méthode de Monte Carlo est une solution efficace.

Introduisons maintenant les différents éléments que nous utiliserons. Tout d'abord, une *matrice stochastique* $P = (P(x, y))_{x,y\in\Omega}$ si ses coefficients sont tous positifs et que la somme des coefficients sur chaque ligne est égale à 1. À partir d'une telle matrice, on peut donner la définition d'une chaîne de Markov.

Définition 1 (Chaîne de Markov)**.** *Soit P une matrice stochastique sur Ω. Une suite de variables aléatoires $(X_n)_{n\geq 0}$ à valeurs dans Ω est appelée chaîne de Markov de matrice de transition P si pour tous $n \in \mathbb{N}$ et $x \in \Omega$, on a*

$$\mathbb{P}[X_{n+1} = x|X_0, \ldots, X_n] = \mathbb{P}[X_{n+1}|X_n] = P(X_n, x)$$

Dans la définition précédente, on remarque que la probabilité conditionnelle de X_{n+1} sachant le passé de la suite de variables aléatoires $(X_k)_{0\leq k\leq n}$ dépend uniquement de la valeur de X_n. On dit alors que les transitions entre deux états se fait sans mémoire pour une chaîne de Markov. Explorons ensemble quelques propriétés sur les chaînes de Markov.

Définition 2 (Irréductible)**.** *Soit $(X_n)_{n\geq 0}$ une chaîne de Markov à valeurs dans Ω ayant P pour matrice de transition. Cette chaîne de Markov est dite irréductible si pour tout élément x,$y \in \Omega$ il existe un entier k tel que $P^k(x, y) > 0$.*

Cette dernière caractéristique peut être interprétée de la manière suivante. Pour une chaîne de Markov irréductible, il est possible de passer depuis n'importe quel état vers n'importe quel autre en un temps fini.

Définition 3 (Apériodique)**.** *Une chaîne de Markov $\mathcal{M}$ est dite apériodique si est seulement si pour tout $x \in \Omega, PGCD\{k : P^k(x, x) > 0\} = 1$*

De plus , la chaîne de Markov $(X_n)_{n\geq 0}$ peut échantillonner selon la distribution π si elle est *ergodique*, c'est à dire que $(X_n)_{n\geq 0}$ a une unique distribution stationnaire π sur Ω tel que $\mathbb{P}[X_t = y|X_0 = x] \to \pi(y)$ quand $t \to \infty$ pour toutes paires d'états $x, y \in \Omega$.

Théorème 4. *Une chaîne de Markov à valeurs dans un espace d'états fini, irréductible et apériodique est aussi ergodique*

Définition 5 (Loi stationnaire)**.** *Soit $(X_n)_{n\geq 0}$ une chaîne de Markov dans Ω de matrice de transition P. Une mesure $(\pi_i)_{i\in\Omega}$ est une loi stationnaire ou probabilité stationnaire si elle vérifie les conditions suivantes :*

- $\pi = \pi P$
- $\forall i \in \Omega, \pi_i \geq 0$
- $\sum_{i\in\Omega} \pi_i = 1$

Définition 6 (Réversible)**.** *Soit $(X_n)_{n\geq 0}$ une chaîne de Markov dans Ω avec π une loi stationnaire. $(X_n)_{n\geq 0}$ est dite réversible par rapport à π si et seulement si :*

$$\pi(x)P(x,y) = \pi(y)P(y,x) \text{ pour tout } x, y \in \Omega \tag{1.1}$$

Lemme 7. *Soit $(X_n)_{n\geq 0}$ une chaîne de Markov dans Ω. S'il existe une loi de probabilité π dans Ω qui vérifie l'équation (1.1), alors π est une loi stationnaire pour $(X_n)_{n\geq 0}$.*

Une loi stationnaire peut être vue comme un vecteur propre à gauche pour une matrice de transition P. Il est donc naturel d'essayer d'étudier les propriétés spectrales de la matrice de transition P. De plus, en imposant que P possède des caractéristiques de réversibilité, on peut voir P comme étant un opérateur auto-adjoint. Avec un espace de produit intérieur adapté, nous pouvons utiliser la théorie spectrale des opérateurs auto-adjoint. Cette démarche est notamment détaillée dans le document [8].

1.1.2 Théorie spectrale

Dans cette partie, (P, π) est irréductible. On peut voir P comme étant un opérateur auto-adjoint sur $L^2(\pi)$ qui est l'espace de produit intérieur pertinent, avec le produit interne définie par :

$$\langle \phi | \psi \rangle = \sum_{x\in\Omega} \frac{\phi(x)\cdot\psi(x)}{\pi(x)}$$

De plus, par algèbre linéaire standard, on sait que P possède $N = |\Omega|$ valeurs propres réelles $1 = \lambda_0 > \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_{N-1} \geq -1$. L'espace $L^2(\pi^{-1})$ admet une base orthonormée constituée des vecteurs propres $v_0 = \pi, v_1, \ldots, v_{N-1}$ de P relatifs aux valeurs propres $\lambda_0, \lambda_1, \ldots, \lambda_{N-1}$.

Ainsi, notre distribution initiale sur Ω peut être réécrite comme $\mu^{(0)} = c_0\pi + c_1 v_1 + \cdots + c_{N-1}v_{N-1}$ avec $c_i = \langle \mu^{(0)}, v_i \rangle$. Après t étapes, la distribution

$$\mu^{(t)} = \mu^{(0)}P^t = \pi + c_1\lambda_1^t v_1 + \cdots + c_{N-1}\lambda_{N-1}^t v_{N-1}. \tag{1.2}$$

D'après ce qui précède, toutes les valeurs propres λ_i pour $1 \leq i \leq N-1$ ont une valeur absolue inférieure à 1. Quand $t \to \infty$, les termes correspondants dans (1.2) deviennent non significatifs, et donc $\mu^{(t)} \to \pi$. D'autre part, pour une chaîne ergodique, l'équation (1.2) démontre également que le taux de convergence vers π est régi par la deuxième plus grande valeur propre en valeur absolue, $\lambda_{\max} = \max\{|\lambda_1|, |\lambda_{N-1}|\}$.

En effet, pour $x \in \Omega$, on pose $P^t(x, \cdot)$ la distribution de l'état de la chaîne de Markov au temps t, dont l'état initial est l'état x.

Définition 8 (Distance en variation)**.** *La distance en variation au temps t depuis l'état initial x est définie comme étant la différence statistique entre la distribution de $P(x, \cdot)$ et $\pi(\cdot)$:*

$$\delta_x(t) = \frac{1}{2} \sum_{y \in \Omega} |P^t(x, y) - \pi(y)|$$

Pour mesurer le taux de convergence, nous utilisons le temps de mélange τ_x qui pour $\epsilon > 0$, est défini par :

$$\tau_x(\epsilon) = \min\{t : \delta_x(t') \leq \epsilon \text{ pour tout } t' \geq t\} \tag{1.3}$$

Généralement, on dit qu'une chaîne de Markov se mélange rapidement si $\tau_x(\epsilon)$ est de l'ordre de $O(\text{poly}(\log(|\Omega|/\epsilon)))$. La proposition ci-dessous précise notre intuition selon laquelle une grande valeur de l'écart spectral $(1 - \lambda_{\max})$ capture la vitesse de convergence vers la loi stationnaire.

Proposition 9. *Soit $(X_n)_{n \geq 0}$ une chaîne de Markov ergodique de matrice de transition P. On a π une loi stationnaire relative à la chaîne de Markov. Le temps de mélange $\tau_x(\epsilon)$ satisfait les inégalités suivantes :*

$$\tau_x(\epsilon) \leq (1 - \lambda_{\max})^{-1} \left(\log \pi(x)^{-1} + \log \epsilon^{-1} \right). \tag{1.4}$$

$$\max_{x \in \Omega} \tau_x(\epsilon) \geq \frac{1}{2} \lambda_{\max} (1 - \lambda_{\max})^{-1} \log(2\epsilon)^{-1}. \tag{1.5}$$

Une preuve de cette proposition peut être trouvé dans [8]. En regardant les expressions de la proposition ci-dessus, on remarque qu'un grand écart $(1 - \lambda_{\max})$ améliore la vitesse de convergence vers la distribution stationnaire. Le choix de l'état initial x intervient dans une moindre mesure, ce qui est tout à fait souhaitable en pratique. De plus, une méthode souvent utilisé en pratique est de rendre la chaîne de Markov *lazy*. L'idée principale est d'ajouter une probabilité de maintien d'un demi à chaque état, c'est à dire de remplacer P par $\frac{1}{2}(I + P)$ avec I la matrice identité de taille $|\Omega| \times |\Omega|$. Cela garantit que toutes les valeurs propres

sont positives tout en réduisant l'écart spectral $(1 - \lambda_{\max})$ seulement d'un facteur 2. Cela nous permet de nous concentrer uniquement sur la valeur propre λ_1 pour borner notre temps de mélange $\tau_x(\epsilon)$.

Pour connaître la valeur de la seconde valeur propre λ_1, plusieurs méthodes sont possibles. Nous allons maintenant en présenter deux : la méthode de couplage et la méthode des chemins canoniques.

1.1.3 Méthode de couplage

Utilisée pour la première fois par Aldous [1], la méthode de couplage est une approche élégante pour borner le temps de mélange sans limiter explicitement l'écart spectral. L'idée derrière le couplage est la suivante. On pose $\mathcal{Y} = (Y_t)_{t\geq 0}$ et $\mathcal{Z} = (Z_t)_{t\geq 0}$ deux processus stochastiques sur l'espace d'états Ω qui sont deux copies de la chaîne de Markov $(X_n)_{n\geq 0}$, dont nous voulons connaître une borne sur le temps de mélange. Notre but est de définir leur évolution conjointe afin de favoriser la coalescence rapide de (Y_t) et (Z_t). Ainsi, de cette manière, nous aurons $X_t = Y_t$ pour un certain t suffisamment grand.

Définition 10 (Couplage)**.** *Soit $(X_n)_{n\geq 0}$ une chaîne de Markov finie et ergodique définie sur l'espace d'états Ω avec P comme matrice de transition. Un couplage est un processus conjoint $(Y, Z) = (Y_t, Z_t)$ sur $\Omega \times \Omega$, tel que chacun des processus Y et Z, considéré marginalement, a la même loi que la chaîne $(X_n)_{n\geq 0}$. Ainsi, on a pour tout $y, y', z, z' \in \Omega$,*

$$\mathbb{P}[Y_{t+1} = y' | Y_t = y, Z_t = z] = P(y, y') \text{ et,}$$
$$\mathbb{P}[Z_{t+1} = z' | Y_t = y, Z_t = z] = P(z, z')$$

Dans la définition précédente, nous n'avons aucune condition sur l'évolution conjointe entre (X_t) et (Y_t). D'où, tout l'intérêt du couplage est de permettre la possibilité que $\mathbb{P}[X_{t+1} = x', Y_{t+1} = y' | X_t = x, Y_t = y] \neq P(x, x')P(y, y')$ pour que X_t et Y_t se rejoignent rapidement.

Conformément aux résultats dans [1], la probabilité que le temps de coalescence dépasse une certaine valeur t pour une distribution π' pour X_0 est une limite supérieure de la distance en variation entre la distribution stationnaire π de $(X_n)_{n\geq 0}$ et la distribution de la chaîne au temps t à partir de la distribution π'.

Lemme 11. *Soit $(X_n)_{n\geq 0}$ une chaîne de Markov finie et ergodique, et soit (Y_t, Z_t) un couplage pour $(X_n)_{n\geq 0}$. Supposons que $\mathbb{P}[Y_t \neq Z_t] \leq \epsilon$, uniformément sur le choix de l'état initial (Y_0, Z_0). Alors le temps de mélange τ_ϵ de la chaîne de Markov est majoré par t, peu importe l'état initial.*

Démonstration. Soit $Y_0 = y$ un état initial arbitraire et soit Z_0 est une variable aléatoire suivant la distribution stationnaire π de $(X_n)_{n\geq 0}$. On pose $A \subseteq \Omega$ un évènement. On a

$$\begin{aligned}\mathbb{P}[Y_t \in A] &\geq \mathbb{P}[Z_t \in A, Y_t = Z_t] \\ &\geq 1 - \mathbb{P}[Z_t \notin A] - \mathbb{P}[Y_t \neq Z_t] \\ &\geq \mathbb{P}[Z_t \in A] - \epsilon \\ &= \pi(A) - \epsilon\end{aligned}$$

Ce qui implique que la distance en variation entre $P^t(\cdot,\cdot)$ et π, que l'on a noté par $\delta_y(t)$ précédemment, est d'au plus ϵ. □

On remarque notamment ici que l'avantage principal est que l'on ne traite pas directement le spectre de la chaîne de Markov. Bien qu'élégante, cette méthode peut parfois être difficile à mettre en oeuvre en pratique. La méthode des chemins canoniques peut être une bonne alternative.

1.1.4 La méthode des chemins canoniques

Cette méthode repose sur le paramètre géométrique de conductance de la chaîne de Markov. Dans cette partie, on suppose que la chaîne de Markov $(X_n)_{n\geq 0}$ est ergodique et que π une loi stationnaire relative à cette chaîne.

Définition 12 (Conductance)**.** *La conductance d'une chaîne de Markov $(X_n)_{n\geq 0}$ est définie de la manière suivante.*

$$\Phi := \min_{S\subset\Omega, 0<\pi(S)\leq 1/2} \frac{Q(S, S^c)}{\pi(S)}, \tag{1.6}$$

où $Q(x,y) = \pi(x)P(x,y) = \pi(y)P(y,x)$, $\pi(S)$ est la densité de probabilité de S sous la distribution stationnaire π de $(X_n)_{n\geq 0}$, et $Q(S,S^c) = \sum_{x\in S, y\in S^c} \pi(x)P(x,y)$.

La conductance peut être considérée comme suit. Pour un ensemble S donné, le quotient de $\frac{Q(S,S^c)}{\pi(S)}$ peut être interprété comme étant la probabilité conditionnelle que la chaîne s'échappe du sous-ensemble S en une seule étape. Par conséquent, Φ permet de mesurer la capacité de $\mathcal{M}$ à s'échapper de n'importe quelle région de l'espace d'état, et donc à progresser rapidement vers la distribution stationnaire. D'où son rapport étroit avec les propriétés de mélange rapide de la chaîne de Markov, qui à son tour est liée à la deuxième valeur propre λ_1 expliqué précédemment. Une preuve de ce théorème peut être trouvée dans [2] [17] [15].

Théorème 13. *La deuxième valeur propre d'une chaîne réversible satisfait :*

$$1 - 2\Phi \leq \lambda_1 \leq 1 - \frac{\Phi^2}{2} \tag{1.7}$$

Corollaire 14. *Soit $(X_n)_{n\geq 0}$ est une chaîne de Markov sur l'espace d'états Ω, réversible et ergodique avec les probabilités $P(x,x) \geq 1/2$ pour tous les états x. On pose Φ comme étant la conductance de $(X_n)_{n\geq 0}$. Alors le temps de mélange de $(X_n)_{n\geq 0}$ vérifient*

$$\tau_x(\epsilon) \leq 2\Phi^{-2}(\ln \pi(x)^{-1} + \ln \epsilon^{-1})$$

Il nous suffit donc d'avoir une bonne borne sur la conductance pour avoir un mélange rapide d'une chaîne de Markov. Pour cela, nous allons utiliser la méthode des chemins canoniques développée dans [12] [17] [16]. L'idée principale est d'essayer d'associer des chemins canoniques entre chaque paire d'états, de telle sorte qu'une transition de la chaîne ne soit pas utilisée par trop de chemins. Le but est donc d'éviter au maximum la présence de "goulot d'étrangement" important qui peut entraver le mélange.

On définit le graphe orienté pondéré $\mathcal{G}((X_n)_{n\geq 0})$ avec l'ensemble des sommets Ω et d'arête e entre une paire ordonnée (x,y) de poids $Q(e) = Q(x,y) = \pi(x)P(x,y)$ chaque fois que $P(x,y) > 0$. On nomme ce graphe le graphe sous-jacent de $(X_n)_{n\geq 0}$.

L'ensemble de chemins canoniques pour $(X_n)_{n\geq 0}$ est un ensemble Γ de chemins $\{\gamma_{xy}\}$ dans le graphe $\mathcal{G}((X_n))$, un entre chaque paire ordonnée (x,y) de sommets distincts. Afin de limiter la conductance, nous souhaitons avoir un ensemble de chemins canoniques qui ne surchargent aucune transition de la chaîne de Markov. Pour mesurer cette « surcharge », nous définissons le paramètre de congestion des chemins ρ pour un ensemble de chemins canoniques Γ comme :

$$\rho(\Gamma) = \max_{e\in\mathcal{G}((X_n));e=(e^-,e^+)} \frac{1}{Q(e)} \sum_{\gamma_{xy}\ni e} \pi(x)\pi(y), \tag{1.8}$$

Si on considère la chaîne de Markov comme un ensemble de flux dans lequel $\pi(x)\pi(y)$ unités de flux se déplacent de x vers y le long du chemin γ_{xy}. De plus, $Q(e)$ est la probabilité que la chaîne de Markov dans la distribution stationnaire utilise la transition $e = (e^-, e^+)$ en une seule étape. Elle sert aussi de capacité de e. La quantité $\rho(\Gamma)$ mesure la surcharge maximale de n'importe quelle arête par rapport à sa capacité. La congestion des chemins $\rho = \rho((X_n))$ de la chaîne $(X_n)_{n\geq 0}$ est définie comme la congestion minimale pouvant être atteinte par tout ensemble de chemins canoniques, autrement dit,

$$\rho = \inf_{\Gamma} \rho(\Gamma)$$

Maintenant que nous avons explicité la notion de congestion liée à la méthode des chemins canoniques, regardons ensemble son lien avec la valeur de la deuxième valeur propre, et donc du temps de mélange de la chaîne de Markov. Celui-ci a été

réalisé pour la première fois par Diaconis et Strook [8], puis amélioré par Sinclair [16] pour notre étude ultérieure.

On modifie la mesure de congestion $\rho(\Gamma)$ pour prendre également en compte les longueurs des chemins. Pour un ensemble $\Gamma = \{\gamma_{xy}\}$ de chemins canoniques, la quantité principale est maintenant :

$$\bar{\rho}(\Gamma) = \max_{e=(e^-,e^+)} \frac{1}{Q(e)} \sum_{x,y;\gamma_{xy}\ni e} \pi(x)\pi(y)|\gamma_{xy}|, \tag{1.9}$$

où $|\gamma_{xy}|$ désigne la longueur du chemin γ_{xy}. Et finalement, le théorème ci-dessous nous donne le lien souhaité entre congestion et le temps de mélange de la chaîne de Markov. Une preuve de ce dernier peut être trouvée dans [16], avec une version plus générale.

Théorème 15. *Pour toute chaîne de Markov réversible et tout choix de chemins canoniques* Γ*, la deuxième plus grande valeur propre* λ_1 *satisfait*

$$\lambda_1 \leq 1 - \frac{1}{\bar{\rho}(\Gamma)}.$$

Nous venons donc d'introduire la méthode des chemins canoniques et expliquer comment elle nous permet d'en déduire une borne sur le temps de mélange d'une chaîne de Markov $(X_n)_{n\geq 0}$. Notre objectif est de trouver un algorithme pour simuler uniformément les polyominos de taille donnée fixée. Une borne sur le temps de mélange de cet algorithme sera proposé grâce à la méthode que nous venons d'expliquer. Mais avant de passer à la déclaration de l'algorithme, nous allons faire une petite introduction sur les polyomino.

1.1.5 Introduction sur les polyominos

Un polyomino (ou animal) est une figure géométrique composée d'une ou plusieurs cellules en forme de carré jointes bord à bord. Une autre manière de définir cet objet est de considérer le maillage carré $\mathbb{Z}^2$. Un polyomino est alors un ensemble d'unité 4-connexe. Le nom polyomino a été inventé par Solomon W. Golomb en 1953 [19] Ces formes ont été popularisées par Martin Gardner dans "Mathematical Games" en novembre 1960 dans Scientific American. Elles sont utilisées dans les puzzles depuis 1907, et sont aussi les structures principales du célèbre jeu vidéo Tetris.

L'énumération des polyominos de taille donnée est aussi une source de recherche. On peut distinguer deux types de polyominos. Les polyominos libres sont

distincts lorsqu'aucun n'est une translation, rotation, réflexion d'un autre. A comparer avec les polyominos fixes qui considèrent comme distincts deux polyominos qui sont rotation ou réflexion de l'autre. Iwan Jensen a énuméré les polyominos fixes jusqu'à $n = 56$ [11] en 2004. Pour l'énumération des polyominos libres, les dernières publications sont faites par John Mason en 2023 qui énumère jusqu'à $n = 50$ [14]. Dans la suite, nous allons nous concentrer uniquement sur les polyominos fixes.

Bien que l'énumération des polyominos soit compliquée, une estimation du nombre de polyominos fixes est donnée en 2000 par I. Jensen et A. Guttman. On pose a_n le nombre de polyominos de taille n. On a :

$$a_n \sim \frac{c\lambda^n}{n} \qquad \text{où } \lambda \simeq 4.0626 \text{ et } c \simeq 0.3169$$

Ce résultat n'est qu'une estimation de l'accroissement du nombre de polyomino en fonction de leur taille. Cependant, seul le résultat théorique suivant a pu être prouvé :

$$\lim_{n\to\infty} (a_n)^{\frac{1}{n}} = \lambda$$

Ainsi, a_n a une croissance exponentielle. Donc, la génération aléatoire de polyomino de grande taille n'est pas triviale. Le premier algorithme a été dévoilé pendant les années 70 par Stauffer [18]. Il repose sur un système de type "remove/insert" des cellules. Cependant aucune vitesse de convergence n'a été établie jusqu'à présent. C'est donc à ce problème que nous allons nous intéresser dorénavant.

1.1.6 Algorithme d'échantillonnage

Dans cette partie, nous allons détailler l'algorithme que nous utilisons pour générer un polyomino de taille donnée, noté n. On peut voir cette algorithme comme la fusion de deux algorithmes.

Tout d'abord, on utilise l'algorithme "remove/insert" pour passer d'un polyomino à un autre. Etant donné un polyomino A, on définit $\bar{A}$ l'ensemble des cellules du maillage $\Omega \setminus A$ qui dispose d'au moins une arête en commun avec une cellule de A. A chaque appel de l'algorithme, on sélectionne uniformément une cellule du polyomino A de départ et on la remplace par une cellule de $\bar{A}$ tiré aléatoirement.

Cependant, en utilisant ce procédé, il est possible que le résultat ne nous donne pas un polyomino. En effet, lorsqu'on retire une cellule, nous pouvons perdre la propriété de connexité indispensable pour que notre ensemble de cellules forme un polyomino. Si on utilise cette méthode dans notre simulation numérique, nous allons faire beaucoup d'itérations qui ne nous serviront pas. On remarque empiriquement que pour exécuter 10000 itérations avec modification du polyomino vers

un autre il nous faut près de 30000 appels à l'algorithme de "remove/insert"(ces tests ont été exécutés avec $n = 50$). Une étude plus complète pourra être faite mais on remarque que cette méthode n'est pas très efficace.

Pour rendre le processus plus efficace, nous allons procéder de la manière suivante. Etant donné un polyomino A, on calcule l'ensemble $\mathbb{V}_A$ des polyominos qui peuvent être construits en ajoutant et retirant une cellule de A. Une autre interprétation est que $\mathbb{V}_A$ regroupe l'ensemble des polyominos accessibles en une étape par l'algorithme de "remove/insert" avec A comme polyomino de départ. C'est dans cet ensemble que nous allons piocher de manière équiprobable à chaque itération pour passer d'un polyomino à un autre. Ainsi, on assure qu'à chaque itération on obtient bien un polyomino. De plus, un autre avantage est qu'avec cette méthode une simple mise à jour suffit pour avoir le nouvel ensemble de polyomino voisin à chaque itérations.

Cependant, dans cette configuration, la chaîne de Markov induite par l'algorithme ne dispose pas des propriétés que l'on souhaite avoir pour réaliser notre étude sur le temps de mélange. Pour remédier à ce problème, nous allons modifier notre algorithme en utilisant l'algorithme de Métropolis-Hastings. Dans celui-ci, au lieu d'accepter toutes les transitions, on calcule un paramètre d'acceptation de la nouvelle transition. Ainsi, à chaque itération la transition proposée peut être acceptée ou non.

On définit par α le paramètre d'acceptation dont son expression est la suivante.

$$\alpha = \frac{\pi(A')Q(A, A')}{\pi(A)Q(A', A)}$$

où π est notre loi cible, la loi uniforme sur l'ensemble de polyominos de taille n, A et A' deux polyominos voisins. On a aussi Q qui peut être vu comme la loi instrumentale qui nous permet de passer vers un autre état selon la méthode expliquée précédemment. Ainsi, $Q(A, A')$ nous donne la probabilité de transition du polyomino A vers le polyomino A', ce qui vaut $\frac{1}{\#\mathbb{V}_A}$. Par conséquent, dans notre cas $\alpha = \frac{\#\mathbb{V}_{A'}}{\#\mathbb{V}_A}$.

On remarque que si le nombre de voisins de A' est supérieur à celui de A, alors l'algorithme de Metropolis-Hasting acceptera à chaque fois la transition. L'algorithme 1.1.6 présente les instructions qui vont nous donner la possibilité de générer des polyominos de taille n de façon uniforme.

Ce pseudo-code nous montre bien que l'on a besoin de mélanger nos polyominos avant d'en avoir un qui soit simulé selon la loi uniforme. Un outil possible pour mesurer le taux de convergence de cet algorithme est le temps de mélange. On définit $\tau_A(\epsilon)$ le temps de mélange pour $\epsilon > 0$ par :

Algorithm 1 Algorithme de génération aléatoire de Polyominos

Require: A un polyomino de taille n.
On calcule $\mathbb{V}_A$ l'ensemble des voisins de A en utilisant "remove/insert
for t allant de 1 à $T_{\max}$ **do**
On prend A' aléatoirement dans $\mathbb{V}_A$.
On calcule $\mathbb{V}_{A'}$ par mise à jour de $\mathbb{V}_A$.
On calcule le paramètre d'acceptation α.
Avec probabilité $\max(\alpha, 1)$:
On remplace A par A' et $\mathbb{V}_A$ par $\mathbb{V}_{A'}$.
end for

$$\tau_A(\epsilon) = \min\left\{ t : \Delta_A(t) := \frac{1}{2} \sum_{\mathbf{B} \in A_n} |P^{t'}(A, B) - \pi(B)| \leq \epsilon \text{ pour tout } t' \geq t \right\}$$

Dans les sections précédentes, nous avons vu que le temps de mélange peut être majoré grâce aux résultats de la Proposition 9. Dans la suite, nous allons montrer que ce temps de mélange peut être majoré par une fonction qui a pour ordre $O(n^{\ln(n)})$. Mais pour cela nous devons regarder plus précisément une caractéristique des polyominos : l'épluchage.

1.2 Épluchage d'un polyomino

1.2.1 Définitions

Pour nous aider à avoir une borne supérieure faible pour notre temps de mélange de l'algorithme 1.1.6, nous allons explorer la notion épluchage pour un polyomnio.

Définition 16 (Epluchage)**.** *Soit $n \geq 1$ et A un polyomino de taille n. L'épluchage pour le polyomino A est une séquence $(A_n, \ldots, A_1)$ tel que :*

— $A = A_n \supset \cdots \supset A_2 \supset A_1$.

— *Pour tout i, A_i est un polyomino de taille i.*

Ainsi, cette caractéristique peut être vue comme étant une séquence de décomposition d'un polyomino. Pour passer entre deux étapes de la séquence, on sélectionne une cellule de A_i notée par c_i qui peut être retirée tout en assurant que l'on garde un polyomino en sortie. Ainsi, nous avons donc $A_i = A_{i-1} \cup c_i$. Une autre manière d'écrire l'épluchage de A est de proposer une séquence $(c_n, \ldots, c_1)$ de l'ensemble des cellules à éliminer pour peler totalement le polyomino. Cette dernière notation est nommée séquence des cellules pelées dans la suite du document.

Prenons un exemple pour clarifier le concept d'épluchage pour un polyomino. Sur la 1.1 nous avons un polyomino de taille 10 que l'on nomme A. On remarque que l'ensemble des cellules $\{C3; D2; \ldots\}$ bien 4-connexe. Une séquence des cellules pelées possibles est par exemple la séquence $\mathcal{D}_1 = \{C3, D2, D3, D4, D5, E6, E5, E4, F4, F3\}$. De plus, la séquence $\mathcal{D}_2 = \{D2, C3, D3, F3, D4, E4, F4, D5, E5, F6\}$ est un autre épluchage possible. On note alors $\mathcal{P}(A)$ l'ensemble des épluchage possible pour le polyomino A.

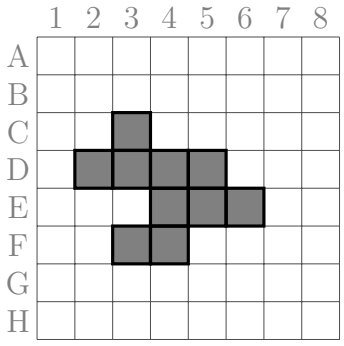

FIGURE 1.1 – Exemple de polyomino de taille 10.

On attire l'attention sur le passage entre l'étape 1 et la 2 pour la séquence s_0. À la première étape, nous supprimons la cellule $C3$. Mais pour garder la propriété

de 4-connexité sur la séquence, nous ne pouvons pas enlever une cellule voisine à la première pour notre deuxième étape. Ainsi, deux cellules qui se suivent dans la séquence n'implique pas qu'elles soient juxtaposées dans le polyomino.

Pour ainsi caractériser les différents épluchages d'un polyomino, nous définissons la fonction de score suivante.

Définition 17. *Soient A et B deux polyominos tels que $B \subset A$. On pose $h(A, B)$ le nombre de composantes 4-connexes de l'ensemble $A \setminus B$. Ainsi, le coût d'épluchage pour un épluchage $\mathcal{D}$ de A est :*

$$h_{\mathcal{D}} := \max_{B \in \mathcal{D}} h(A, B)$$

Sur la figure 1.2, on a en gris foncé un sous-ensemble de cellules formant un polyomino B potentiellement présenté dans un épluchage de A. Lors de cette étape, nous avons donc $h(A, B) = 3$.

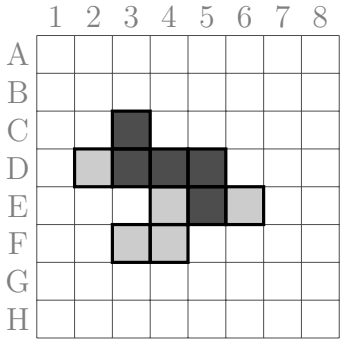

FIGURE 1.2 – Représentation d'un épluchage possible

Et plus généralement, on peut définir le coût d'épluchage optimal pour un polyomino A par :

$$H(A) := \min_{\mathcal{D} \in \mathcal{P}(A)} \max_{B \in \mathcal{D}} h(A, B)$$

Et enfin de la même manière, on peut considérer H_n le coût d'épluchage global maximum pour tous les polyominos de taille n.

$$H_n := \max_{A \in A_n} H(A)$$

Il s'avère que ces notions nous permettront de réduire la borne supérieure du temps de mélange pour notre algorithme 1.1.6. Regardons plus en détails comment se comporte H_n en fonction de n.

1.2.2 Borne supérieure sur H_n

Dans cette partie, nous allons rechercher principalement une borne supérieure sur H_n. On pose

$$\alpha := \left(\ln\left(\frac{1}{\beta}\right)\right)^{-1} \tag{1.10}$$

où β est le nombre positif réel qui satisfait $\beta^4 + \beta - 1 = 0$. Ce qui nous donne une valeur approchée pour $\alpha \sim 3,1$. La justification du choix des paramètres α et β est expliquée dans la suite du document. Grâce à ces derniers, nous avons le résultat suivant.

Théorème 18. *Pour tout $n \geq 2$, nous prouvons la borne supérieure sur le coût d'épluchage H_n des polyominos de taille n suivante :*

$$H_n \leq \alpha \ln(n) \tag{1.11}$$

Ainsi, le coût d'épluchage global croît de façon logarithmique en fonction de n. Cependant, la preuve de ce résultat nécessite un développement détaillé sur la notion d'épluchage des polyominos. Il est possible d'associer les épluchages d'un polyomino à un graphe orienté.

L'arbre d'épluchage

Définition 19 (Arbre d'épluchage)**.** *Soit A un polyomino de taille n et $\mathcal{D} \in \mathcal{P}(A)$ un épluchage de A. On considère $\mathcal{D}$ par sa séquence des cellules pelées $(c_n, \ldots, c_1)$. Un arbre d'épluchage T pour A est un graphe orienté où l'ensemble des sommets est donné par les cellules $\{c_n, \ldots, c_1\}$ et l'ensemble des arêtes est défini de la manière suivante :*

- *Pour tout indice $n \geq i > 1$, c_i a un seul voisin sortant qui est le premier dans la séquence $(c_{i-1}, \ldots, c_1)$ admettant une 4-connexité avec c_i.*
- *c_1 n'admet aucun voisin sortant.*

Vérifions ensemble l'intégrité de cette définition. Après avoir enlevé les cellules $c_n, \ldots, c_{i+1}$, l'ensemble restant est toujours 4-connexe (par définition des séquences d'épluchage). La cellules c_i a donc bel et bien au moins un voisin 4-connexe dans l'ensemble $\{c_{i-1}, \ldots, c_1\}$.

Un arbre d'épluchage est donc associé à un épluchage mais plusieurs épluchages d'un polyomino peuvent former le même arbre d'épluchage. D'autres propriétés peuvent directement être données.

Lemme 20. *Soit A un polyomino de taille n, $(c_n, \ldots, c_1)$ un épluchage de A et son arbre d'épluchage associé T.*

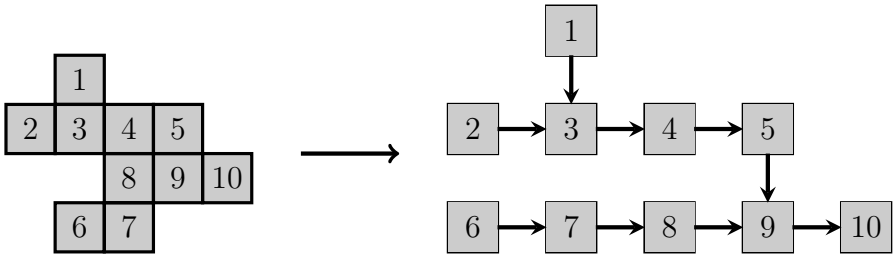

FIGURE 1.3 – A gauche, un polyomino A de taille 10 et son épluchage $(1, \ldots, 10)$ dans lequel les cellules sont représentées par des nombres $1, \ldots, 10$. A droite, l'arbre d'épluchage associé. Remarquons que l'arête sortant de la cellule 4 est 5 et non 8 car, parmi les cellules restantes 4-connectées 5 et 8 à la cellule 4, la cellule 5 est la première à être supprimée.

— *Le graphe orienté T est un arbre enraciné en c_1 où tous les sommets ont au plus trois arêtes entrantes et un sommet sortant, à l'exception de c_1 qui possède au plus quatre arêtes entrantes et zéro sortante.*
— *Si on pose pour tout i, T_i le sous-arbre de T enraciné en c_i, alors l'union des cellules constituant le sous-arbre T_i est un polyomino.*

Maintenant concentrons-nous sur les épluchages qui se terminent par une cellule donnée et les conséquences sur les différentes propriétés associés.

Condition supplémentaire sur un épluchage

Etant donné un polyomino A et une cellule c de A, on note $\mathcal{P}(A, c)$ l'ensemble des épluchages de A se terminant par c. On peut ainsi considérer le coût d'épluchage optimal pour un polyomino A se terminant par la cellule c :

$$H(A, c) := \min_{D \in \mathcal{P}(A,c)} \max_{B \in D} h(A, B) \ .$$

Il est facile de voir que quelque soit la cellule c d'un polyomino, on a $H(A) \leq H(A, c)$. En effet, cela se justifie par le fait que $\mathcal{P}(A, c)$ est inclus dans $\mathcal{P}(A)$. Le lemme suivant nous informe de combien il est possible de dépasser la valeur optimal quand on impose un choix sur la dernière cellule d'une séquence de d'épluchage.

Lemme 21. *Soit A un polyomino et c une cellule de A ayant au plus trois cellules 4-connectées dans A. Alors,*

$$H(A, c) \leq H(A) + 2 \ . \tag{1.12}$$

Démonstration. Soient A un polyomino et $\mathcal{D} = (c_n, \ldots, c_1)$ un épluchage optimal

tel que son coût est $H(A)$. Notons T l'arbre d'épluchage associé à cette séquence, avec c_1 comme cellule enracinée. On considère c une autre cellule de A.

Dans l'arbre T, il y a un unique chemin orienté de c vers c_1. Notons ce chemin $\tau = (c, \ldots, c_1)$. Inversons maintenant l'orientation de ce chemin. Le graph orienté résultant est noté $\bar{T}$. Cet arbre est composé de l'ensemble des sommets $\{c_n, \ldots, c_1\}$ et est enraciné en c (cf figure 1.4).

FIGURE 1.4 – À gauche : un arbre d'épluchage T avec j comme cellule enracinée. Le chemin τ entre les cellules c et j est entouré en rouge. À droite : l'arbre $\bar{T}$ résultant après avoir inversé les arêtes de τ.

Détaillons ensemble un épluchage $\bar{D}$ du polyomino A dont l'arbre d'épluchage associé est $\bar{T}$ et qui satisfait :

$$\max_{B \in \bar{D}} h(A, B) \leq H(A) + 2 \ .$$

Puisque $\bar{D} \in \mathcal{P}(A, c)$, on aura donc $H(A, c) \leq H(A) + 2$ ce qui conclura notre preuve.

On sait que la cellule racine c_1 a au plus quatre voisins entrants dans l'arbre d'épluchage T et au moins un. Notons T_1, T_2, T_3, T_4 les sous-arbres de T dont les racines sont les quatre cellules voisines de c_1. Si c_1 admet moins de quatre voisins entrants alors les T_i peuvent être vides. Sans perte de généralité, on suppose que T_1 contient le chemin τ. De plus, on sait que l'épluchage optimal D se termine par la suppression de la cellule c_1. Donc pour réaliser cette dernière étape, les polyominos A_1, A_2, A_3, A_4 correspondant respectivement aux sous-arbres T_1, T_2, T_3, T_4 doivent être supprimés au préalable intégralement. Puisque D est un épluchage optimal, le coût d'épluchage de chaque A_i est inférieur à $H(A)$

Commençons l'épluchage $\bar{D}$. On débute par la suppression des trois polyominos A_2, A_3 et A_4. Grâce à la remarque du précédent paragraphe, le pelage de ces trois polyominos a un coût inférieur à $H(A)$ chacun. Si on commence par la suppression totale de A_2, alors celle de A_3 a un coût inférieur à $H(A)+1$. Cette hausse provient de la composante 4-connexe A_2 déjà supprimée. Et enfin, la suppression totale de A_4 a un coût inférieur à $H(A) + 2$ où le terme '+2' correspond aux composantes 4-connectées supprimées A_2 et A_3. Dès lors, nous pouvons finalement supprimer la

cellule c_1. On peut remarquer que cette dernière suppression rassemble l'ensemble des cellules pelée et forme ainsi qu'une seule composante connexe.

Reproduisons maintenant le même processus sur la cellule sortante de c_1 dans $\bar{T}$, notons la $c_{(2)}$. Par hypothèse, $c_{(2)}$ a au plus trois voisins entrants dans l'arbre d'épluchage $\bar{T}$. On distingue même deux cas : soit $c_{(2)}$ est en fait la racine c de $\bar{T}$ et, par hypothèse du lemme, a au plus trois voisins entrants, soit $c_{(2)}$ n'est pas la racine de $\bar{T}$ et admet alors un voisin sortant et au plus trois voisins entrants. Désignons donc par $T_2(2), T_3(2), T_4(2)$ les sous-arbres de $\bar{T}$ dont les racines sont des voisins entrants de $c_{(2)}$ (ces sous-arbres peuvent potentiellement être vide). Sans perte de généralité, on suppose que $T_2(2)$ contient la cellule c_1 précédemment supprimée. On pose $A_i(2)$ le polyomino correspondant à $T_i(2)$. Ainsi avec cette notation, on a déjà $A_2(2)$ qui est supprimé par l'épluchage $\bar{D}$. Puisque $T_3(2)$ et $T_4(2)$ sont aussi des sous-arbres de l'arbre d'épluchage T associés à l'épluchage optimal D, le coût d'épluchage de $A_3(2)$ et de $A_4(2)$ est au plus de $H(A)$ chacun. Comme pour le cas de A_2,A_3 et A_4, nous pouvons éplucher $A_3(2)$ et $A_4(2)$ pour un coût maximum de $H(A)+2$. Après la suppression de ces deux ensembles, nous pouvons supprimer la cellule $c_{(2)}$ pour nous ramener une nouvelle fois à un ensemble de cellules pelées qui ne forme plus qu'une composante connexe. On répète donc ce processus pour toutes les cellules du chemin τ jusqu'à l'élimination de la cellule cible c. Cela nous donne donc un épluchage $\bar{D}$ de $\mathcal{P}(A,c)$ dont le coût est inférieur à $H(A)+2$.

□

Choix des paramètres α et β

Notre démonstration du Théorème 18 (voir ci-dessous) nécessite que les paramètres α et β satisfassent les deux inégalités suivantes

$$\alpha \ln(\beta n) + 1 \leq \alpha \ln(n) \tag{1.13}$$

et

$$\alpha \ln((1-\beta)n) + 4 \leq \alpha \ln(n)\ . \tag{1.14}$$

L'inégalité (1.13) est équivalente à $\beta \leq e^{-1/\alpha}$ tandis que l'inégalité (1.14) est équivalente à $\beta \geq 1 - e^{-4/\alpha}$. De plus, les quantités $e^{-1/\alpha}$ et $1-e^{-4/\alpha}$ sont respectivement croissant et décroissant en α. On minimise donc α en choisissant l'abscisse à laquelle ces deux courbes se croisent et pour β, l'ordonnée correspondante :

$$\beta = e^{-1/\alpha} = 1 - e^{-4/\alpha}\ . \tag{1.15}$$

Ceci explique le choix fait dans (1.10). En particulier, puisque $x^4 + x - 1$ est une fonction croissante, on vérifie facilement que $\beta > \frac{\sqrt{2}}{2}$ puis $\alpha > \frac{2}{\ln(2)}$.

Nous pouvons maintenant passer à la preuve du Théorème 18. L'idée principale est de réaliser cette preuve par récurrence. L'inégalité (1.11) est vraie pour $n = 2$ puisque $H_2 = 1$ et $\alpha > \frac{2}{\ln(2)}$. Soit $n \geq 3$ un entier. On suppose que (1.11) est vrai pour tout entier $k < n$. Soit A un polyomino de taille n. On cherche à prouver que $H(A)$ est plus petit que $\alpha \ln(n)$.

Soit $\mathcal{T}$ la collection de tous les sous-arbres des arbres d'épluchage de A. Pour tout $T \in \mathcal{T}$, on appelle taille de T, notée $\#T$, le nombre de sommets dans T. On choisit alors $T_0 \in \mathcal{T}$ comme suit :

$$\text{Parmi les sous-arbres } T \in \mathcal{T} \text{ avec } \#T \leq \frac{n}{2}, T_0 \text{ a une taille maximale}$$

Par exemple, dans la Figure 1.3, le sous-arbre composé de cellules $\{1, \ldots, 5\}$ a une taille maximale (i.e. $\frac{n}{2} = 5$) et pourrait alors jouer le rôle de T_0.

Dans la suite, on note A_0 l'union des cellules dans T_0. D'après le Lemme 20 sur les propriétés des arbres d'épluchage, A_0 est un ensemble 4-connexe et est donc un polyomino de taille $\#A_0 = \#T_0$. Soit également c_0 la cellule racine de T_0.

Cas 1 : $\#T_0 \geq \lfloor (1-\beta)n \rfloor + 1$, où $\lfloor x \rfloor$ désigne la partie entière du nombre réel x. Soit A' le polyomino obtenu à partir de A lorsque toutes les cellules de T_0 ont été supprimées, c'est-à-dire $A' := A \backslash A_0$. En particulier, $\frac{n}{2} \leq \#A' \leq \lfloor \beta n \rfloor$.
Procédons de la façon suivante pour peler le polyomino A.

1. Nous épluchons d'abord complètement A_0 selon un épluchage optimal se terminant à la cellule c_0 (la racine de T_0). Cela a un coût de pelage $H(A_0, c_0)$ qui est inférieur à $H(A_0)+2$ d'après le Lemme 21. De plus, par construction, la cellule c_0 a un sommet sortant et au plus trois sommets entrants.
2. On décortique alors complètement A' de façon optimale, ce qui conduit à un coût de décorticage d'au plus $H(A') + 1$, où le terme '+1' correspond à l'ensemble 4-connexe A_0 qui a déjà été enlevé et compte comme une composante connexe.

Le processus décrit précédemment nous donne une séquence d'épluchage pour A. Ainsi,

$$H(A) \leq \max\{H(A_0) + 2, H(A') + 1\} \ .$$

Et avec l'hypothèse de récurrence et le fait que $m \mapsto H_m$ est une fonction croissante, on peut écrire

$$H(A_0) + 2 \leq H_{\lfloor \frac{n}{2} \rfloor} + 2 \leq \alpha \ln\left(\frac{n}{2}\right) + 2$$

qui est plus petit que $\alpha \ln(n)$ puisque $\alpha > \frac{2}{\ln(2)}$. De la même manière, en utilisant 1.13 :

$$H(A') + 1 \leq H_{\lfloor \frac{n}{2} \rfloor} + 1 \leq \alpha \ln\left(\frac{n}{2}\right) + 1 \leq \alpha \ln(n) \ .$$

Cas 2 : $\#T_0 \leq \lfloor (1-\beta)n \rfloor$**.** Pour ce cas ci, nous devons d'abord introduire quelques notations. Soit T un arbre à décortiquer de A dont T_0 est un sous-arbre. On note c^* le voisin sortant de c_0. Par maximalité sur la taille de T_0, on obtient l'énoncé suivant dont la preuve est reportée à la fin de la section.

Proposition 22. *L'arbre d'épluchage T admet deux sous-arbres T_1 et T_2 (différents de T_0) dont les cellules racines sont voisines entrantes de c^* et telles que :*
(i) $\max\{\#T_1, \#T_2\} \leq \#T_0 \leq \lfloor (1-\beta)n \rfloor$ *;*
(ii) $\#T_0 + \#T_1 + \#T_2 + 1 > \frac{n}{2}$.

On note par ailleurs que les sous-arbres T_1 ou T_2 peuvent être vides mais pas simultanément (grâce à l'item (ii) de la Proposition 22). Pour $i = 0, 1, 2$, on pose A_i le polyomino correspondant à l'union des cellules de T_i. Notons A^* l'union des cellules de T_0, T_1, T_2 et c^* et $A' := A \backslash A^*$. Par construction, A^* et A' sont bien des polyominos.

Maintenant, procédons à l'épluchage du polyomino A de la manière suivante.

1. On décortique d'abord complètement A_0 selon un décorticage optimal se terminant par c_0. D'après le Lemme 21, cette étape a un coût d'épluchage $H(A_0, c_0)$ inférieur ou égal à $H(A_0) + 2$.
2. On décortique ensuite complètement A_1 selon un épluchage optimal se terminant par sa racine c_1. Cela conduit à un coût d'épluchage de $H(A_1, c_1)+1$ (le terme '+1' correspond à l'ensemble 4-connexe A_0 qui a déjà été supprimé) qui est inférieur à $H(A_1) + 3$ par le lemme 21.
3. On décortique maintenant A_2 (s'il existe) selon un épluchage optimal se terminant à sa racine c_2. Comme précédemment, cette étape correspond à un coût d'épluchage d'au plus $H(A_2, c_2) + 2 \leq H(A_2) + 4$ (cette étape n'existe pas si A_2 est vide).
4. Nous supprimons c^*. Par conséquent, le polyomino entier A^* a été supprimé et compte alors comme une seule composante connexe.
5. Enfin, nous épluchons complètement A' de manière optimale, ce qui conduit à un coût d'épluchage d'au plus $H(A') + 1$, où le terme '+1' correspond à la suppression de A^*.

L'ensemble du processus décrit précédemment est un épluchage possible de A. Ainsi,

$$H(A) \leq \max\{H(A_0) + 2, H(A_1) + 3, H(A_2) + 4, H(A') + 1\} .$$

Avec la Proposition 22, item (i), nous avons d'abord :

$$\begin{aligned} \max\{H(A_0) + 2, H(A_1) + 3, H(A_2) + 4\} &\leq H_{\lfloor (1-\beta)n \rfloor} + 4 \\ &\leq \alpha \ln((1-\beta)n) + 4 \\ &\leq \alpha \ln(n) \end{aligned}$$

par (1.14) et hypothèse de récurrence. Par contre, si on utilise cette fois la Proposition 22, item (ii), on obtient :

$$H(A') + 1 \leq H_{\lfloor \frac{n}{2} \rfloor} + 1 \leq \alpha \ln\left(\frac{n}{2}\right) + 1 \leq \alpha \ln(n) \ .$$

Dans les deux cas, $H(A)$ est plus petit que la limite supérieure recherchée $\alpha \ln(n)$. Ceci conclut la preuve du Théorème 18.

Retour sur la preuve de la Proposition 22

Considérons T_1 et T_2 définis comme deux sous-arbres de l'arbre d'épluchage T dont les cellules racines c_1 et c_2 sont des sommets entrants de c^*, mais différents de T_0. A ce stade, T_1 et T_2 sont éventuellement vides : si T_i est vide alors c_i n'existe pas.

Montrons que $\#T_1 \leq \frac{n}{2}$. Notons par π le chemin de l'arbre d'épluchage T de c^* à la racine de T, disons c (dans le cas où $c^* = c$, π se réduit à $\{c^*\}$). Modifions l'arbre d'épluchage T comme suit : inversez les directions des arêtes de π et aussi celle entre c^* et c_1. Soit T' l'arbre résultant : T' est un arbre à éplucher dont la racine est c_1. Soit T^* le sous-arbre de T' enraciné à c^*. Son complément fixé dans T' correspond aux sommets de T_1. Si, par absurde, on suppose que $\#T_1 > \frac{n}{2}$ alors $\#T^* \leq \frac{n}{2}$. De plus, on a $\#T^* > \#T_0$ puisque T^* contient T_0 et la cellule c^*. Cela contredit la maximalité de T_0.

Nous avons alors prouvé que $\#T_1 \leq \frac{n}{2}$. Par maximalité de T_0, cela force $\#T_1 \leq \#T_0$. De la même manière, on obtient $\#T_2 \leq \#T_0$. L'index (i) est prouvé.

Pour obtenir l'index (ii), nous considérons le sous-arbre $\bar{T}$ de l'arbre d'épluchage T enraciné à c^*. Alors $\#\bar{T} = \#T_0 + \#T_1 + \#T_2 + 1$. Si $\#\bar{T} \leq \frac{n}{2}$ alors $\bar{T}$ est un sous-arbre de T contredisant la maximalité de T_0. Forcément, $\#\bar{T} > \frac{n}{2}$.

1.2.3 Borne inférieure sur H_n

Nous venons donc de terminer la preuve de la borne supérieure. Néanmoins, pour avoir une idée critique sur cette borne, il est important d'avoir aussi une borne inférieure sur H_n. En effet, si la borne supérieure est du même ordre que la borne inférieur, la qualité de cette borne supérieure sera améliorée.

Proposition 23. *Pour tout $n \geq 4^2$, on a une borne inférieure sur le coût d'épluchage globale :*

$$H_n \geq \frac{\ln(n)}{\ln(2)} - 2$$

Démonstration. Construisons par récurrence une famille de polyominos $(F_k)_{k\geq 1}$ ayant une structure fractale et une fonction de score qui s'élève à $H(F_k)$. Voir Figure 1.5.

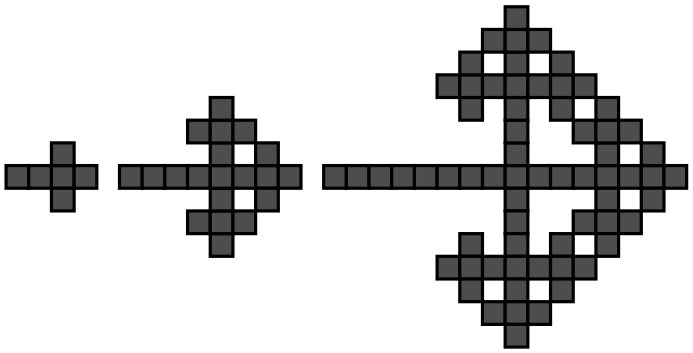

FIGURE 1.5 – De gauche à droite, sont représentés les polyominos F_1, F_2 et F_3 dont les tailles sont resp. 6, 20 et 63, et dont les diamètres sont resp. 4, 8 et 16.

Expliquons comment construire F_{k+1} à partir de F_k. Notons $\mathrm{diam}(F_k)$ et appelons *diamètre*, le nombre de cellules de F_k situées sur la ligne horizontale délimitée par ses cellules les plus à gauche et à droite (incluses). Appelons le *pied* la cellule la plus à gauche de F_k. Nous assemblons d'abord trois copies de F_k par leur pied (deux d'entre elles étant tournées avec un angle $\frac{\pi}{2}$ et $-\frac{\pi}{2}$). On ajoute ensuite une ligne horizontale composée de $\mathrm{diam}(F_k)$ cellules. Ainsi,

(i) $\#F_{k+1} = 3(\#F_k - 1) + 1 + \mathrm{diam}(F_k)$,
(ii) $\mathrm{diam}(F_{k+1}) = 2\,\mathrm{diam}(F_k)$.

Dans F_{k+1}, l'ajout d'une ligne composée de $\mathrm{diam}(F_k)$ cellules garantit que lors de la construction de F_{k+2}, les trois copies de F_{k+1} ne se chevauchent qu'en leurs pieds.

Puisque $\mathrm{diam}(F_1) = 4$ et $\#F_4 = 203 \leq 4^4$, il est facile de prouver que $\mathrm{diam}(F_k) = 2^{k+1}$ pour tout k et $\#F_k \leq 4^k$ pour tout $k \geq 4$.

Lemme 24. *Pour tout $k \geq 2$, on a $H(F_k) = 2k$.*

Effectivement, nous pouvons prouver ce résultat par récurrence sur $k \geq 2$.

$$H(F_k) = 2k \text{ and } H(F_k, c_k) = 2k + 1 \tag{1.16}$$

où c_k est le pied du polyomino F_k. Vérifions d'abord (1.16) pour $k = 2$. Considérons le polyomino $F_1 \setminus \{c_1\}$ de taille 5 représentant une croix ; on peut facilement remarquer que son coût d'épluchage est de 3. Pour éplucher F_2 en terminant en c_2, il faut d'abord éplucher les trois croix et ensuite la ligne horizontale. Cela signifie que $H(F_2, c_2) = 5$. Considérons maintenant l'épluchage de F_2 dans lequel on pèle d'abord deux croix, puis la ligne horizontale et enfin la croix restante. Cela entraîne un coût de pelage égal à 4. Comme cet épluchage est l'optimal, on a $H(F_2) = 4$.

Supposons maintenant que (1.16) est vrai pour $k \geq 2$ et démontrons cette propriété pour $k + 1$. Dans F_{k+1}, appelons la *cellule centrale* la cellule à laquelle les trois copies de F_k sont collées ensemble. Cette cellule centrale est 4-connectée à quatre polyominos : trois copies de $F_k \setminus \{c_k\}$ et une ligne horizontale. Pour obtenir des épluchages optimaux conduisant à $H(F_{k+1})$ et $H(F_{k+1}, c_{k+1})$, il faut choisir parmi ces quatre polyominos les trois qui seront supprimés avant de supprimer la cellule centrale.

Supprimons tout d'abord les trois copies de $F_k \setminus \{c_k\}$. L'épluchage de la première copie a un coût $2k + 1$ par l'hypothèse de récurrence (1.16), l'épluchage du second a un coût $(2k + 1) + 1$ et celui du troisième un coût $(2k + 1) + 2$. Il reste ensuite à supprimer la cellule centrale et la ligne horizontale menant à c_{k+1}. On obtient $H(F_{k+1}, c_{k+1}) = 2k + 1 + 2 = 2(k + 1) + 1$.

Maintenant, choisissons de supprimer d'abord deux copies de $F_k \setminus \{c_k\}$ et la ligne horizontale avant la cellule centrale. Le meilleur choix est de peler la ligne horizontale en dernier. Ce processus a un coût d'épluchage égal à $2k + 2$. Il reste ensuite à supprimer la cellule centrale, et terminer par la troisième copie de $F_k \setminus \{c_k\}$. Lors de cette dernière étape, le coût de pelage ne dépasse pas $2k + 2$. Donc $H(F_{k+1}) = 2(k + 1)$.

□

Nous pouvons maintenant terminer la preuve de la Proposition 23. Pour $k \geq 2$, le Lemme 24 nous dit que :

$$H_{4^k} \geq H_{\#F_k} = \max_{A \in \mathcal{A}_{\#F_k}} H(A) \geq H(F_k) = 2k \ .$$

Étant donné $n \geq 4^2$, soit k l'entier tel que $4^k \leq n < 4^{k+1}$. En utilisant à nouveau le fait que $m \mapsto H_m$ est croissant, on peut écrire :

$$H_n \geq H_{4^k} \geq 2k \geq \frac{\ln(n)}{\ln(2)} - 2 \ .$$

1.3 Borne sur le temps de mélange

Nous venons de faire une exploration sur la propriété d'épluchage. Cette dernière nous permet d'avoir une meilleure borne supérieure sur le temps de mélange de notre algorithme d'échantillonnage.

Proposition 25. *Soit $(A_t)_{t\in\mathbb{N}}$ la chaîne de Markov induite par l'algorithme de Metropolis-Hastings avec la Proposition 1 et des polyominos de taille n. Alors, nous avons la borne supérieure sur le temps de mélange suivante :*

$$\tau_A(\epsilon) \leq 14n^{4\lceil \alpha \ln(n)\rceil+12}\left(\ln a_n + \ln \epsilon^{-1}\right)$$

Pour avoir ce résultat, nous utilisons la méthode des chemins canoniques développée dans la Section 1.1. Dans l'expression de la mesure de congestion modifiée, $\gamma(A,B) = \{A = A_0; A_1, A_2, \ldots, A_K = B\}$ est un ensemble d'arêtes pour aller de A à B où A_i est un polyomino et (A_i, A_{i+1}) représente la transition entre deux polyominos avec une seule cellule de différence. On note par Γ un ensemble de tous les chemins $\gamma(A,B)$ entre les polyominos A et B.

Détaillons maintenant le processus qui nous permet de passer d'un polyomino donné à un autre. L'idée principale est que quelque soit le polymino A, il existe un chemin vers n'importe quel autre polyomino B. Cette transition est toujours composée de n étapes. Le polyomino B est construit cellule par cellule en ajoutant les premières cellules de B à droite de A (en commançant par une cellule à l'extrême gauche du polyomino B), et en continuant ainsi vers la droite pas à pas en remontant une séquence d'épluchage minimisant, le plus possible, le coût d'épluchage.

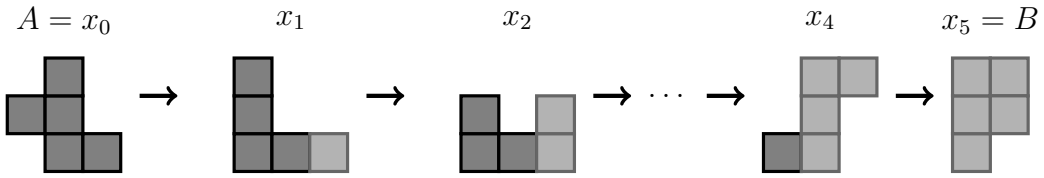

FIGURE 1.6 – Exemple de processus de transition depuis A (tout à gauche) vers B (tout à droite).

Prenons un exemple pour rendre plus concret le processus. Sur la Figure 1.6, nous avons une mise en évidence des étapes de transition entre le polyomino A et le polyomino B. Cette transition est bien composée de cinq étapes (cinq étant la taille des deux polyominos). À chaque étape, on supprime une cellule provenant du polyomino A, en gris foncé, et l'on ajoute une autre qui complétera la partie droite du polyomino dans l'objectif de reformer B, en gris clair.

Ce processus est bien réalisable par notre algorithme car nous modifions seulement une seule cellule à la fois. En utilisant la mesure de congestion modifiée $\bar{\rho}(\Gamma)$, qui avec le théorème 15 et la proposition 9 nous permet de borner le temps de mélange, on obtient :

$$\begin{aligned}\bar{\rho}(\Gamma) &= \max_{e=(e^-,e^+)} \frac{1}{\pi(e^-)P(e^-,e^+)} \sum_{A,B;e\in\gamma(A,B)} \pi(A)\pi(B)|\gamma(A,B)| \\ &\leq \max_{e=(e^-,e^+)} \frac{n}{a_n P_n(e^-,e^+)} \sum_{A,B;e\in\gamma(A,B)} 1\end{aligned}$$

où nous rappelons que a_n représente le nombre de polyominos de taille n. Maintenant, nous limitons $P_n(e^-, e^+)$, la version paresseuse (*"lazy"*) de la matrice de transition initiale :

$$P(e^-,e^+) = q(e^-,e^+)\Big(1 \wedge \frac{q(e^-,e^+)}{q(e^+,e^-)}\Big) \geq \frac{1}{6n^2}$$

Cette valeur s'explique de la manière suivante. À chaque étape, nous avons une chance sur deux de ne pas modifier le polyomino donné (ce qui n'est pas utilisé en pratique pour des soucis de performance). Puis une cellule est tirée au hasard dans e^- pour être déplacé vers sa position suivante qui est choisie parmi l'ensemble des cellules voisines. Cet ensemble voisin compte un maximum de 3 voisins par cellule du polyomino. Alors,

$$\bar{\rho}(\Gamma) \leq \max_{e=(e^-,e^+)} \frac{6n^3}{a_n} \sum_{A,B;e\in\gamma(A,B)} 1 \tag{1.17}$$

Maintenant, nous nous concentrons sur la délimitation du nombre de termes dans la somme de l'inégalité (1.17). Étant donnée une transition $e = (e^-, e^+)$, il y a une séparation verticale comme représenté sur la Figure 1.7 entre A en cours d'épluchage et B en cours de construction. Nous ne savons pas où cette séparation se trouve, mais il y a un maximum de n possibilités pour cela. Selon l'endroit où elle se trouve, la taille de la partie gauche, noté par k, sera comprise entre 1 et n. La partie droite est naturellement de taille $n - k$.

Cela signifie que pour récupérer A depuis e^- nous devons compléter la partie gauche avec $n - k$ cellules et la partie droite de e^+ avec $k - 1$ cellules. Si on ne considère que la partie gauche de e^-, d'après le Lemme 21, comme on impose que la dernière cellule à supprimer soit une des cellules les plus à droite, il existe un nombre maximum de H_n+2 composantes connectées qui ont été supprimées et qu'il convient ensuite d'ajouter pour récupérer A. Par Théorème 18, pour un polyomino de taille n, ce nombre est majoré par : $\lceil \alpha \ln(n) \rceil + 2$. De plus, ces composantes ont un

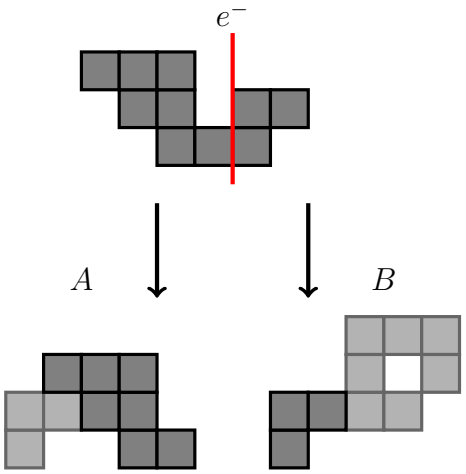

FIGURE 1.7 – Exemple de séparation verticale quand A et B sont connus.

maximum de $3k$ possibilités de se connecter sur e^-. En utilisant un raisonnement similaire sur e^+, nous pouvons déduire la borne suivante :

$$\sum_{A,B;e\in\gamma(A,B)} 1 \leq \sum_{k=1}^{n-1} \left(\sum_{i=1}^{\lceil \alpha \ln(n) \rceil + 2} \binom{3k}{i} \sum_{k_1+\cdots+k_i=n-k} a_{k_1} \dots a_{k_i} \right) \cdot \left(\sum_{j=1}^{\lceil \alpha \ln(n) \rceil + 2} \binom{3(n-k)}{j} \sum_{k_1+\cdots+k_j=k} a_{k_1} \dots a_{k_j} \right) \tag{1.18}$$

On réécrit maintenant la majoration de la manière suivante :

$$\begin{aligned} \sum_{A,B;e\in\gamma(A,B)} 1 \leq & a_n \sum_{k=1}^{n-1} \left(\sum_{i=1}^{\lceil \alpha \ln(n) \rceil + 2} \binom{3k}{i} \sum_{k_1+\cdots+k_i=n-k} 1 \right) \cdot \left(\sum_{j=1}^{\lceil \alpha \ln(2) \rceil + 2} \binom{3(n-k)}{j} \sum_{k_1+\cdots+k_j=k} 1 \right) \\ & \text{car } a_k a_{n-k} \leq a_n \\ = & a_n \sum_{k=1}^{n-1} \left(\sum_{i=1}^{\lceil \alpha \ln(n) \rceil + 2} \binom{3k}{i} \binom{n-k-1}{i-1} \right) \cdot \left(\sum_{j=1}^{\lceil \alpha \ln(n) \rceil + 2} \binom{3(n-k)}{j} \binom{k-1}{j-1} \right) \end{aligned}$$

en utilisant un résultat de la somme d'une série géométrique sur le nombre de partitions de respectivement $n-k$ et k. Ainsi :

$$
\begin{aligned}
\sum_{A,B;e\in\gamma(A,B)} 1 \leq & a_n \sum_{k=1}^{n-1} \left(\sum_{i=0}^{\lceil \alpha \ln(n) \rceil + 2} (3k(n-k))^i \right)^2 \\
= & a_n \sum_{k=1}^{n-1} \left(\frac{(3k(n-k))^{\lceil \alpha \ln(n) \rceil + 3} - 1}{3k(n-k) - 1} \right)^2 \\
= & a_n \sum_{k=1}^{n-1} \left(\frac{(3k(n-k))^{\lceil \alpha \ln(n) \rceil + 3}}{2k(n-k)} \right)^2 \\
\leq & \frac{9}{4} a_n \sum_{k=1}^{n-1} (3k(n-k))^{2\lceil \alpha \ln(n) \rceil + 4} \\
\leq & \frac{9}{4} n^{4\lceil \alpha \ln(n) \rceil + 9} a_n. \qquad (1.19)
\end{aligned}
$$

Maintenant, en revenant à l'inégalité de Poincaré et en injectant l'inégalité (1.19) dans l'inégalité (1.17), on obtient :

$$\bar{\rho}(\Gamma) \leq 14 n^{4\lceil \alpha \ln(n) \rceil + 12}.$$

Et finalement, on obtient :

$$\tau_A(\epsilon) \leq 14 n^{4\lceil \alpha \ln(n) \rceil + 12} \left(\ln \pi(A)^{-1} + \ln \epsilon^{-1} \right).$$

1.4 Simulation numérique

Nous avons obtenu une borne supérieure sur le temps de mélange pour notre algorithme de Metropolis-Hasting. Ainsi, nous avons une idée du nombre d'itérations dont nous avons besoin pour être sûr que nous générons bien des polyominos de taille donnée de façon uniforme. Il est possible de faire une critique sur notre résultat en faisant une simulation numérique.

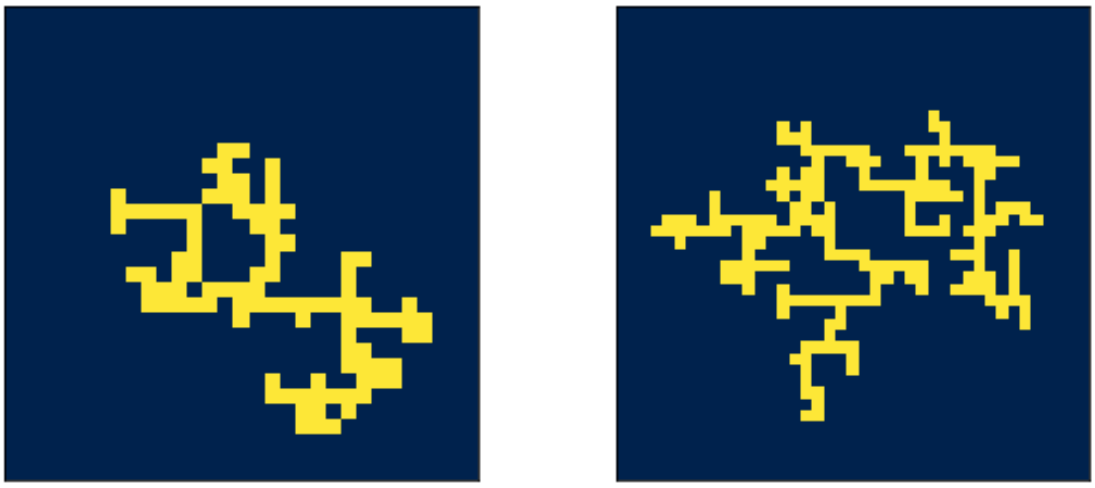

FIGURE 1.8 – Deux exemples de polyominos, de taille 100 à gauche et de taille 200 à droite.

Nous allons programmer l'algorithme (1.1.6) en python. En plus, nous allons faire une sauvegarde des polyominos à chaque itération. Cela va nous permettre de comparer les différents polyominos explorés par rapport à notre polyomino de départ. Pour cela, nous allons utiliser une distance de recouvrement pour différencier deux polyominos. Etant donné deux polyominos A et B, $d(A, B)$ a pour valeur le nombre minimum de cellules différentes lorsque l'on superpose les deux polyominos à translation près.

Avec les graphes de la figure 1.9, on remarque que notre borne supérieure théorique peut être surévaluée. Selon ce critère, on remarque que les distances entre deux polyominos commencent à se stabilité plus tôt. En effet, pour les polyominos de taille 100, la distance par rapport au polyomino initial se stabilise à partir d'environ 1000 itérations. Et pour les polyominos de taille 200, la stabilisation de la distance commence à partir de 3000 itérations de l'algorithme de Metropolis-Hasting.

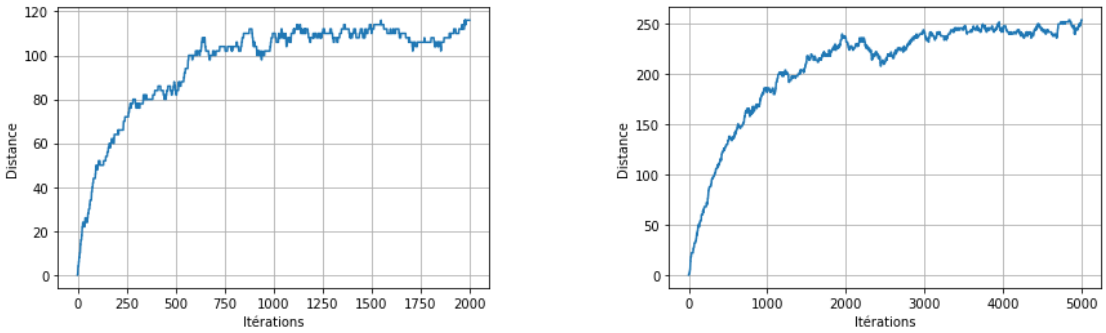

FIGURE 1.9 – Distance par rapport au polyomino de départ à chaque itération, pour des polyominos de taille 100 à gauche et de taille 200 à droite.

Par conséquent, si l'on considère que le critère de la distance par recouvrement nous permet de différencier la proximité des polyominos, alors on peut dire que notre borne sur le temps de mélange semble surévalué. Un temps de mélange de l'ordre de cn, avec c une constante, pourrait nous suffire pour que notre algorithme nous génère des polyominos de façon uniforme.

1.5 Application à la ville de Lille

Nous allons maintenant appliquer notre étude sur la ville de Lille du XVIIIème siècle. L'objectif est de proposer une simulation en nous aidant d'une génération de polyomino aléatoire sur le maillage de la ville. Ces polyominos serviront de base pour définir les mouvements des personnages de notre simulation. Nous détaillerons d'ailleurs dans cette partie comment il est possible de passer d'un ensemble de polyominos à un ensemble de cycles, qui définiront les points de passage des personnages.

Une source importante des données pour notre application provient du plan-relief. Il s'agit d'une représentation en trois dimensions d'une ville fortifiée (en l'occurrence la ville de Lille) inscrit dans un environnement élargie de son territoire. Aujourd'hui, les plan-reliefs désignent des objets historiques presque exclusivement en France et au Canada.

Le plan-relief de la Figure 1.10 a été réalisé par l'ingénieur Nicolas de Nézot entre le mois d'août 1740 et le 20 juin 1743. Avec une dimensions de 9.80 par 6.70 mètres, il est l'un des plus grands lors de sa création. Hélas, le plan-relief de Lille est amputé, vers 1900, d'une partie de ses tables de campagne. Cette mésaventure a réduit la taille de ce dernier à 4,40 par 4 mètres.

De nos jours, les plan-reliefs sont une source importante de données pour les historiens. Plusieurs travaux de recherche ont été réalisés sur ces objets, dont [7] qui a été une bonne source d'informations. On peut retrouver de nombreux plan-reliefs dans les musées modernes.

Dans ce document, nous allons nous concentrer uniquement sur la ville intérieure de Lille délimitée par la muraille. La Figure 1.12 est une capture d'une vue du dessus du plan-relief centré sur la ville intérieure. On remarque bien avec ce point de vue que la ville est délimitée par les douves (en vert clair). On peut aussi remarquer les axes routiers principaux. Sur la Figure 1.13, on a mis en évidence les routes principales de la ville facilement visible sur le plan-relief. Ainsi, l'ensemble des routes forment un maillage qui va nous aider à générer des polyominos aléatoires dans la ville de Lille.

Pour mieux comprendre les mouvements possibles à l'intérieur de l'enceinte, nous allons modifier notre ensemble de polyominos générés en un autre ensemble de cycles aléatoires. D'ailleurs, en augmentant progressivement le nombre de polyominos générés, nous pourrons déceler si des arêtes sont plus souvent utilisées. Ces arêtes en question donneront des indices concernant les rues où le risque d'embouteillage est le plus élevé pour des futures études.

FIGURE 1.10 – Plan-relief de la ville de Lille. *Source : https ://pba.lille.fr/Collections/Chefs-d-OEuvre/Plans-Reliefs/Plan-relief-de-Lille/*

On distingue 135 "quartiers" sur le plan de la Figure 1.14. Pour générer un polyomino dans cet environnement, nous commençons par sélectionner aléatoirement l'une des cellules avec une loi uniforme. Puis, on ajoute une par une les cellules parmi l'ensemble des cellules 4-connexes voisines jusqu'à obtenir la taille voulue. Ce qui nous donne notre polyomino initial. À partir de ce dernier, on peut utiliser l'algorithme 1.1.6 pour avoir la quantité de polyominos souhaités pour notre maillage de la ville de Lille.

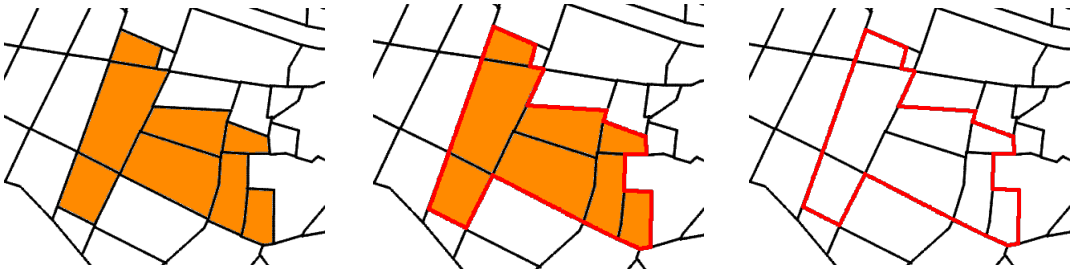

FIGURE 1.11 – Transformation du polyomino vers un cycle

Pour chaque polyomino généré, nous pouvons créer un cycle qui lui est associé. En posant A un polyomino, le cycle $\mathcal{C}_A$ qui lui est associé correspond à l'ensemble des arêtes sur le périmètre extérieur. Il est possible que, dans la liste, il y ait des polyominos qui possèdent un ou plusieurs trous en leur centre. Dans ce cas, l'ensemble des arêtes est composé de plusieurs composantes connexes. Le cycle est alors un sous-ensemble de l'ensemble des arêtes composé d'une seule composante

connexe et de taille maximale. Plus formellement, on pose E_A l'ensemble des arêtes qui ne séparent pas deux cellules de A. Le cycle $\mathcal{C}_A$ associé au polyomino A est l'ensemble des arêtes défini par :

$$\mathcal{C}_A := \max_p \{e_1; \ldots; e_p | \mathcal{C}_A \text{ est un ensemble d'arêtes avec une seule composante connexe}\}$$

Nous avons maintenant un ensemble de cycles provenant de la génération aléatoire de polyomino. On peut notamment remarquer que pour deux polyominos de même taille donnée leur cycle n'est par forcément de la même longueur.

FIGURE 1.12 – Vue de dessus du plan-relief de la ville de Lille

FIGURE 1.13 – Mise en évidence des routes principales facilement visible du plan-relief de la ville de Lille

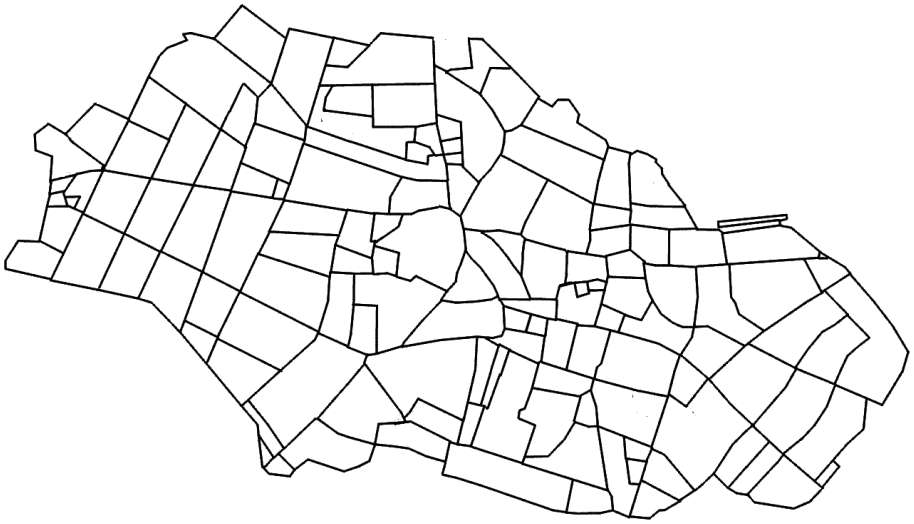

FIGURE 1.14 – Maillage de la ville de Lille

FIGURE 1.15 – Quatre exemples de polyominos dans la ville de Lille

Chapitre 2

Échantillonnage d'équipes

Dans cette partie, nous allons rechercher une méthode d'échantillonnage d'un sous-ensemble de taille fixée en fonction de leur score. En s'appuyant sur les principes développés dans la section précédente pour une chaîne de Markov, nous allons construire un algorithme Metropolis-Hasting qui nous permet de simuler n'importe quelle loi π. Dans notre cas, le score de chaque sous-ensemble sera donné par un système d'apprentissage, comme un réseau de neurones. Enfin, une borne sur le temps de mélange introduit plus tôt sera aussi proposée.

Tout au long de ce chapitre, nous allons motiver notre étude avec l'utilisation du contexte du jeu vidéo. Un environnement particulièrement bien adapté est celui des jeux d'arène de bataille en équipe, ou MOBA (multiplayer online battle arena) comme Dota 2, League of Legends, etc. Pour ce style de jeux vidéo, avoir des adversaires proposant une certaine difficulté peut être très intéressant. À partir d'un algorithme d'apprentissage, une intelligence artificielle peut être conçue pour les personnages du jeux. Cependant, cela reste une tâche difficile, surtout lorsque l'on dispose d'une puissance de calcul modeste.

Une piste pour aider la performance de l'apprentissage est de l'adapter en fonction de la puissance potentielle des adversaires ou des équipes. Généralement dans les *MOBA*, une liste de héros est proposée aux joueurs. Cette liste est composée de héros plus ou moins puissants en fonction de leur rôle, de la composition des équipes et des objectifs des éditeurs du jeu. Pour *Dota 2*, les joueurs peuvent choisir parmi une liste de 124 héros. Par construction et par choix sur le développement du jeu, une équipe de cinq héros donnée peut avoir peu de chance de victoire contre une autre équipe composé de héros disposant d'une forte synergie. Ainsi, une méthode qui permet d'améliorer les performances des systèmes d'apprentissage est de proposer régulièrement à une équipe d'entrainement un certain

type d'équipe adverse. Si une équipe adverse dispose de synergies importantes et qu'elle a tendance à être forte contre l'équipe d'entrainement, alors la phase d'entrainement peut être plus difficile pour être compétitive. C'est à ce moment que notre algorithme d'échantillonnage peut aider.

L'objectif ici est de sélectionner aléatoirement des paires d'équipes en priorisant ceux dont leurs résultats de match ne sont pas nuls. Ainsi, l'IA aura tendance à être entraînée dans les cas les plus difficiles. À titre d'étude de cas, nous considérerons le jeu vidéo Dota 2 dans la suite de notre développement.

Pour simplifier notre étude, nous considérons deux problèmes qui peuvent être résumés en un seul problème d'échantillonnage. Deux équipes, chacune avec $k = 5$ joueurs, s'affrontent dans le jeu *Dota 2*. Une particularité de ce jeu est qu'un joueur doit choisir un héros parmi $n = 124$ héros et aucun héros ne peut être choisi plus d'une fois au sein des deux équipes.

Tout d'abord, regardons ensemble le problème d'inférence suivant, $z = s(x)$ où z est une probabilité de gagner pour une équipe x représentée par un sous-ensemble de k héros parmi n choix possible. On représente l'entrée x par n valeurs binaires, où chaque valeur indique si le héros correspondant est présent ou non. Dans notre seconde partie, l'entrée est doublée, n entrées pour une équipe et n autres pour l'équipe adverse. Le problème d'inférence est alors noté $z = s(x, y)$ avec x et y les deux équipes adverses représentées par deux sous-ensembles. L'importance du second cas est motivée puisque les résultats dépendent certainement des deux compositions.

Nous pouvons remarquer que si nous souhaitons échantillonner juste pour maximiser le score, un réseau contradictoire génératif (GAN) [9] serait suffisant car les GAN résolvent précisément ce problème. Cependant, ici nous ne souhaitons pas spécialement maximiser le score mais simplement privilégier les matchs à score élevé.

2.1 Échantillonnage d'une équipe

2.1.1 Introduction

Pour notre premier cas, notre idée est d'échantillonner une équipe aléatoire selon $\pi(x) \propto s(x)$, où $s(x)$ est la sortie d'un réseau de neurones correspondant au score de l'équipe x. On note alors $\pi(x) = \frac{s(x)}{Z}$ où Z est la constante de normalisation. Soit Ω l'ensemble des équipes possibles de cinq héros dans le jeu *Dota 2*. On pose $(X_n)_{n\geq 0}$ notre chaîne de Markov ergodique à valeur dans Ω et qui admet π comme loi stationnaire. On sait que le nombre de combinaison de toutes les équipes possibles est $\binom{n}{k}$ de sorte que $Z \geq \binom{n}{k} s_{\min}$ où $s_{\min}$ désigne le score minimal.

Pour connaître le score de chaque équipe, on utilise une base de données contenant un historique de matches joués [6]. On peut remarquer ici que les scores de chaque paire d'équipes sont liées puisque lorsqu'une équipe gagne alors elle marque 1, l'adversaire marque 0 de sorte que la moyenne des scores est égal à $0,5$, soit :

$$\frac{1}{|\Omega|} \sum_{x \in \Omega} s(x) = \frac{1}{2}$$

où Ω représente toutes les équipes possibles composées de 5 joueurs pris parmi les héros à $n = 124$. Pour ce qui concerne l'algorithme d'échantillonnage, nous utilisons l'algorithme de Metropolis-Hastings [10] en utilisant pour loi de proposition $Q(x, y)$ défini par la méthode suivante :

— On pose x une équipe donnée.
— On choisit un héros au hasard parmi les $k = 5$ héros actuels x.
— On remplace le héros sélectionné par un autre pris uniformément parmi les $n - k$ héros restant. On note cette nouvelle équipe y.
— On calcul le taux d'acceptation α comme suit :

$$\alpha = 1 \wedge \frac{\pi(y)Q(y, x)}{\pi(c)Q(x, y)},$$

On a donc ici x et y sont deux équipes identiques à l'exception d'un seul héros. Comme expliqué dans le premier chapitre, un outil possible pour mesurer le taux de convergence est le temps de mélange de τ_ϵ défini pour $\epsilon > 0$ par :

$$\tau_x(\epsilon) \; = \; \min\{t : \Delta_x(t') \leq \epsilon \text{ for all } t' \geq t\}.$$

Au lieu d'utiliser la notion de congestion vu précédemment, nous allons cette fois-ci utiliser une méthode proche de celle des chemins canoniques pour avoir notre borne supérieur sur le temps de mélange $\tau_x(\epsilon)$.

2.1.2 Méthode des flux multiproduits

Cette méthode est une généralisation naturelle qui capture exactement les temps de mélange, du même ordre la conductance. L'idée principale, développée par Sinclair dans [16], est de répartir le flux sur le chemin γ_{xy} entre une paire (x, y) d'états sur plusieurs chemins. On reprend la notation $\mathcal{G}((X_n)_{n\geq 0})$ pour désigner un réseau de flux dans lequel une unité de flux doit être acheminée de x à y pour chaque paire ordonnée (x, y) de sommets distincts, et chaque arête orientée e a une *capacité* notée $Q(e)$. À la différence des chemins canoniques, cette méthode nous autorise de diviser le flux entre x et y en plusieurs chemins. Cela nous permet de rechercher un *flux multiproduits fractionnaire* qui minimise la congestion. Ainsi, vu la similitude vis à vis de la méthode des chemins canoniques, nous aurons donc une limite similaire sur le temps de mélange. Passons maintenant aux différentes définitions.

Définition 26 (fonction flux)**.** *Un flux dans $\mathcal{G}((X_n)_{n\geq 0})$ est une fonction $f : \mathscr{P} \to \mathcal{R}^+$ qui satisfait*

$$\sum_{p\in\mathscr{P}_{xy}} f(p) = 1 \quad \text{pour tout } x, y \in X, x \neq y,$$

où $\mathscr{P}_{xy}$ est l'ensemble de tous les chemins orientés de x à y dans $\mathcal{G}((X_n)_{n\geq 0})$

La qualité du flux est mesurée par le paramètre de congestion $\mathscr{R}(f)$, défini de façon analogue à 1.8 par

$$\mathscr{R}(f) := \max_e \frac{1}{Q(e)} \sum_{x,y} \sum_{\substack{p\in\mathscr{P}_{xy}\\ p\ni e}} \pi(x)\pi(y)f(p). \tag{2.1}$$

De même, on définit le paramètre de congestion allongée $\bar{\mathscr{R}}(f)$ en rajoutant les longueurs des chemins :

$$\bar{\mathscr{R}}(f) := \max_e \frac{1}{Q(e)} \sum_{x,y} \sum_{\substack{p\in\mathscr{P}_{xy}\\ p\ni e}} \pi(x)\pi(y)f(p)|p|. \tag{2.2}$$

Et finalement, nous pouvons faire le lien avec la deuxième valeur propre de la matrice de transition P de notre chaîne de Markov $(X_n)_{n\geq 0}$. Ce qui nous donnera notre borne sur le temps de mélange.

Théorème 27. *Pour toute chaîne de Markov réversible, et pour tout flux f, la deuxième plus grande valeur propre λ_1 satisfait*

$$\lambda_1 \leq 1 - \frac{1}{\mathscr{R}(f)} \quad \text{et} \quad \lambda_1 \leq 1 - \frac{1}{\mathscr{R}(f)l(f)}$$

où $l(f)$ est la longueur du plus long chemin p avec $f(p) > 0$.

2.1.3 Borne sur le temps de mélange

Maintenant notre objectif est de trouver une borne supérieure sur le temps de mélange pour notre problème d'échantillonnage d'équipe pour le jeu *Dota 2*. Notre point de départ est la définition du paramètre de congestion $\mathscr{R}(f)$ dont nous devons lui trouver une borne supérieure.

$$\mathscr{R}(f) = \max_{e=(e^-,e^+)} \frac{1}{\pi(e^-)P(e^-,e^+)} \sum_{x,y} \sum_{\substack{p\in\mathscr{P}_{xy}\\ p\ni e}} \pi(x)\pi(y)f(p). \tag{2.3}$$

Pour les deux équipes x et y, nous définissons un ensemble de chemins de la manière suivante. Tout d'abord, il faut remarquer que le nombre d'arêtes sur le chemin dépend de $i = |x \triangle y|$, qui représente le nombre de héros différents entre x et y. Ici, les chemins $\mathscr{P}_{xy}$ sont choisis en remplaçant les héros $i = |x\Delta y|$ différents entre x et y dans n'importe quel ordre possible, ce qui donne donc $i!$ différents chemins que nous choisissons avec une probabilité équivalente. Avec cette réflexion, on pose f le flux équiprobable, où pour tout $p \in \mathscr{P}_{xy}$, $f(p) = 1/i!$ pour tout x et y dans Ω différents.

À partir de la description de ces premiers éléments, on peut prouver le résultat de la proposition suivante.

Proposition 28. *Si $k^2 \leq n$, l'inégalité de Poincaré nous donne la borne supérieure suivante :*

$$\mathscr{R}(f) \leq \frac{s_{\max}^3}{s_{\min}^3} k(n-k)e^4.$$

Démonstration. Premièrement, pour toute arête e entre deux états, on partitionne les chemins $\mathscr{P}_{xy}$ traversant e selon le nombre de héros qui diffère entre x et y.

Réécriture du paramètre de congestion

Le nombre de ces chemins traversant une arête $e = (e^-, e^+)$ (c'est-à-dire avec $e \in p$) est limité par :

$$\sum_{i=1}^{k} \sum_{j=0}^{i-1} \binom{k}{i} \binom{n-k-j}{i-j} \cdot \binom{i}{j} \binom{n-k}{j} \tag{2.4}$$

En effet, pour un p donné, on considère qu'il existe i héros différents entre x et y, donc on a $\binom{k}{i}$ possibilités de sélectionner ces héros parmi x. De plus, nous supposons également que j héros de y ont déjà été choisis dans e^- parmi les i héros

qui sont différents, ce qui donne $\binom{i}{j}$ possibilités. Nous avons au plus $\binom{n-k}{j}$ façons de choisir les héros manquants pour récupérer x et $\binom{n-k-j}{i-j}$ façons de choisir les héros manquants pour obtenir un y, car les k héros dans e^- sont exclus de ce choix et les héros j sont également exclus car déjà choisis pour récupérer x.

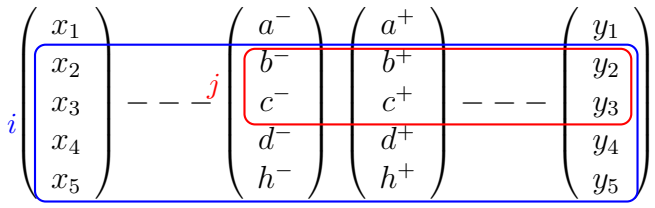

FIGURE 2.1 – Exemple de transition entre deux équipes. De gauche à droite, on a les équipes x, e^-, e^+ et y.

Dans l'exemple de la Figure 2.1, nous avons $i = 4$ héros différents entre x et y. Si l'on considère que $j = 2$ héros ont déjà été choisis dans e^- comme dans y, nous avons encore $i - j = 2$ héros qu'il faut modifier pour récupérer l'équipe x.

Nous venons de voir comment réécrire la double somme de l'expression (2.3). Maintenant, nous allons borner P pour pouvoir avoir notre borne supérieure sur $\mathscr{R}(f)$.

$$\begin{aligned} P(e^-, e^+) =& Q(e^-, e^+) \left(1 \wedge \frac{\pi(e^+)Q(e^+, e^-)}{\pi(e^-)Q(e^-, e^+)}\right) \\ =& Q(e^-, e^+) \wedge \frac{\pi(e^+)Q(e^+, e^-)}{\pi(e^-)} \\ \geq& \frac{s_{\min}}{s_{\max}} \frac{1}{k} \frac{1}{n-k} \end{aligned}$$

où $s_{\min}$ et $s_{\max}$ désignent respectivement le score minimum et le score maximum renvoyés par notre processus d'apprentissage. Par conséquent, en notant que les poids des chemins des couples (x, y) ayant une différence de i héros est $f(p) = \frac{1}{i!}$,

on obtient :

$$\begin{aligned}
\mathscr{R}(f) \leq & \max_e \frac{1}{\pi(e^-)P(e^-,e^+)} \sum_{x,y} \sum_{\substack{p\in\mathscr{P}_{xy}\\ p\ni e}} \pi(x)\pi(y)f(p) \\
\leq & \max_e \frac{1}{\pi(e^-)P(e^-,e^+)} \sum_{i=1}^{k}\sum_{j=0}^{i-1} \binom{k}{i}\binom{n-k-j}{i-j}\binom{i}{j}\binom{n-k}{j}\pi_{\max}^2.\frac{1}{i!} \\
\leq & \frac{\pi_{\max}^2}{\pi_{\min}}\frac{s_{\max}}{s_{\min}}k(n-k)\sum_{i=1}^{k}\sum_{j=0}^{i-1}\frac{1}{i!}\binom{k}{i}\binom{n-k-j}{i-j}\binom{i}{j}\binom{n-k}{j} \\
\leq & \pi_{\max}\frac{s_{\max}^2}{s_{\min}^2}k(n-k)\sum_{i=1}^{k}\sum_{j=0}^{i-1}\frac{1}{i!}\binom{1}{i}\binom{n-k-j}{i-j}\binom{i}{j}\binom{n-k}{j}
\end{aligned}$$

Maintenant, nous pouvons réutiliser la borne supérieure de $\pi_{\max}$ et la borne inférieure de Z définies précédemment.

$$\begin{aligned}
\mathscr{R}(f) \leq & \frac{1}{Z}\frac{s_{\max}^3}{s_{\min}^2}k(n-k)\sum_{i=1}^{k}\sum_{j=0}^{i-1}\frac{1}{i!}\binom{k}{i}\binom{n-k-j}{i-j}\binom{i}{j}\binom{n-k}{j} \\
\leq & \frac{s_{\max}^3}{s_{\min}^3}k(n-k)\sum_{i=1}^{k}\sum_{j=0}^{i-1}\frac{1}{\binom{n}{k}}\frac{1}{i!}\binom{k}{i}\binom{n-k-j}{i-j}\binom{i}{j}\binom{n-k}{j} \qquad (2.5)
\end{aligned}$$

Majoration de la double somme

Nous montrons maintenant que la double somme peut être bornée par une constante. Tout d'abord, nous développons les coefficients binomiaux :

$$\begin{aligned}
&\sum_{i=1}^{k}\sum_{j=0}^{i-1}\frac{1}{\binom{n}{k}}\frac{1}{i!}\binom{k}{i}\binom{n-k-j}{i-j}\binom{i}{j}\binom{n-k}{j} \\
&\quad = \sum_{i=1}^{k}\sum_{j=0}^{i-1}\frac{k!(n-k)!}{n!}\frac{1}{i!}\frac{k!}{i!(k-i)!}\frac{(n-k-j)!}{(i-j)!(n-k-i)!}\cdot \\
&\qquad \frac{i!}{j!(i-j)!}\frac{(n-k)!}{j!(n-k-j)!} \\
&\quad = \sum_{i=1}^{k}\sum_{j=0}^{i-1}\frac{k!k!}{(i-j)!j!i!(k-i)!j!(i-j)!}\cdot\frac{(n-k)!(n-k)!}{n!(n-k-i)!} \\
&\quad = \sum_{i=1}^{k}\sum_{j=0}^{i-1}\frac{k!k!}{(i-j)!j!i!(k-i)!j!(i-j)!}\cdot\frac{(n-k)...(n-k-i+1)}{n...(n-k+1)}
\end{aligned}$$

De plus, on remarque que pour la deuxième fraction il y a k facteurs au dénominateur et i facteurs au numérateur. Ensuite, comme les valeurs du dénominateur

sont supérieures ou égales à $n-k+1$ et que les valeurs du numérateur sont inférieures ou égales à $n-k$, nous pouvons découper la deuxième fraction en fractions toutes inférieures à 1. Ainsi,

$$\begin{aligned}
&\sum_{i=1}^{k}\sum_{j=0}^{i-1}\frac{1}{\binom{n}{k}}\frac{1}{i!}\binom{k}{i}\binom{n-k}{i-j}\binom{i}{j}\binom{n-k}{j} \\
&\qquad\leq\sum_{i=1}^{k}\sum_{j=0}^{i-1}\frac{k!k!}{(i-j)!j!i!(k-i)!j!(i-j)!}\frac{1}{n...(n-k+i+1)} \\
&\qquad=\sum_{i=1}^{k}\frac{k!k!}{i!(k-i)!}\frac{1}{n...(n-k+i+1)}\sum_{j=0}^{i-1}\left(\frac{1}{j!(i-j)!}\right)^2 \\
&\qquad=\sum_{i=1}^{k}\frac{k\dots(i+1)\cdot k\dots(k-i+1)}{n\dots(n-k+i+1)}\sum_{j=0}^{i-1}\left(\frac{1}{j!(i-j)!}\right)^2 \\
&\qquad=\sum_{i=1}^{k}\frac{k\dots(i+1)\cdot k\dots(k-i+1)}{n\dots(n-k+i+1)}\frac{1}{(i!)^2}\sum_{j=0}^{i-1}\left(\frac{i!}{j!(i-j)!}\right)^2 \\
&\qquad\leq\sum_{i=1}^{k}\frac{k\dots(i+1)\cdot k\dots(k-i+1)}{n\dots(n-k+i+1)}\frac{2^{2i}}{(i!)^2} \\
&\qquad=\sum_{i=1}^{k}\frac{k\dots(i+1)\cdot k\dots(k-i+1)}{n\dots(n-k+i+1)}\frac{4^i}{(i!)^2}
\end{aligned}$$

Maintenant, on distingue deux cas, soit $k-i<i$ soit $k-i\geq i$.

Cas 1 : $k-i<i$**.** Avec cette hypothèse, il y a plus d'entiers dans la séquence $k\dots(k-i+1)$ que dans l'autre séquence $k\dots(i+1)$. On obtient donc :

$$\begin{aligned}
&\sum_{i=1}^{k}\frac{k\dots(i+1)\cdot k\dots(k-i+1)}{n\dots(n-k+i+1)}\frac{4^i}{(i!)^2} \\
&\qquad=\sum_{i=1}^{k}\frac{k\dots(i+1)\cdot k\dots(i+1)\cdot i\dots(k-i+1)}{n\dots(n-k+i+1)\cdot i!}\frac{4^i}{i!} \\
&\qquad\leq\sum_{i=1}^{k}\frac{(k\dots(i+1))^2}{n\dots(n-k+i+1)\cdot}\frac{4^i}{i!}
\end{aligned}$$

De plus, nous avons supposé que $k^2\leq n$, et par conséquent nous avons :

$$(k-j)^2\leq n-j\quad\forall j\in\{0,\dots,k\}$$

Nous pouvons donc l'utiliser pour obtenir une limite supérieure à la somme précédente.

$$\sum_{i=1}^{k} \frac{(k \dots (i+1))^2}{n \dots (n-k+i+1)\cdot} \frac{4^i}{i!} \leq \sum_{i=1}^{k} \frac{4^i}{i!} \leq e^4$$

Cas 2 : $k - i \geq i$**.** Dans ce cas, il y a plus d'entiers dans la séquence $k \dots (i+1)$ que dans l'autre séquence $k \dots (k-i+1)$.

$$\begin{aligned}\sum_{i=1}^{k} &\frac{k \dots (i+1) \cdot k \dots (k-i+1)}{n \dots (n-k+i+1)} \frac{4^i}{(i!)^2} \\ &\leq \sum_{i=1}^{k} \frac{(k \dots (i+1))^2}{n \dots (n-k+i+1)\cdot} \frac{4^i}{(i!)^2}\end{aligned}$$

Conclusion

Et, en utilisant notre supposition que $k^2 \leq n$, on obtient :

$$\sum_{i=1}^{k} \frac{(k \dots (i+1))^2}{n \dots (n-k+i+1)\cdot} \frac{4^i}{(i!)^2} \leq \sum_{i=1}^{k} \frac{4^i}{(i!)^2} \leq \sum_{i=1}^{k} \frac{4^i}{i!} \leq e^4$$

Ainsi, nous pouvons injecter cette majoration dans l'expression (2.5) et obtenir

$$\mathscr{R}(f) \leq \frac{s_{\max}^3}{s_{\min}^3} k(n-k)e^4$$

ce qui conclut la preuve □

Ainsi, nous avons la borne suivante sur le temps de mélange pour notre algorithme d'échantillonnage.

$$\tau_x(\epsilon) \leq \frac{s_{\max}^3}{s_{\min}^3} k(n-k)e^4 \left(\ln \pi(x)^{-1} + \ln \epsilon^{-1}\right)$$

On remarque que cette borne dépend des scores que nous retourne notre réseau de neurones. Si les scores d'équipes sont très déséquilibrés, on remarque que notre borne est plus haute. En pratique, la différence entre le score maximum et le score minimum ne dépasse pas 0.8.

2.2 Échantillonnage des doubles équipes

Passons maintenant à l'échantillonnage des équipes doubles. On note le problème d'inférence par $z = s(x, y)$ où x et y sont maintenant deux équipes de $k = 5$ héros et z est la probabilité de gagner pour l'équipe x contre l'équipe y. Évidemment, on a $s(x, y) = 1 - s(y, x)$.

Dans ce contexte, on a maintenant $\pi(x, y) = \frac{s(x,y)}{Z}$ où $s(x, y)$ correspond au score du match entre les équipes x et y. Ensuite, nous avons $Z \geq \binom{n}{2k} s_{\min}$ avec la même limite inférieure pour $s_{\min}$. On pose $(X_n)_{n\geq 0}$ la chaîne de Markov ergodique à valeur dans Ω . Une nouvelle fois, on utilise la méthode des flux multiproduit (2.3) pour nous donner la proposition suivante.

Proposition 29. *Si $k^2 \leq n - 2k$, la méthode des flux multiproduits nous donne la borne supérieure suivante :*

$$\mathscr{R}(f) \leq \frac{s_{\max}^3}{s_{\min}^3} 2k(n-k)e^8$$

Démonstration. Le déroulement de cette preuve est similaire à celle de notre section précédente (proposition 28).

Tout d'abord, nous partitionnons les chemins $\mathscr{P}_{xy}$ traversant e en fonction du nombre de héros qui diffèrent entre (x, y) et (x', y'), noté i, où $i = i_1 + i_2$ avec i_1 et i_2 indiquant respectivement les différences pour la première et la deuxième équipe. Nous considérons que j héros de y ont déjà été choisis dans e^-. Le nombre de ces chemins traversant une arête $e = (e^-, e^+)$ est limité par :

$$\sum_{i_1=0}^{k} \sum_{j_1=0}^{i_1-1} \sum_{i_2=0}^{k} \sum_{j_2=0}^{i_2-1} \binom{k}{i_1}\binom{n-k-j_1}{i_1-j_1}\binom{i_1}{j_1}\binom{n-k}{j_1} \cdot \binom{k}{i_2}\binom{n-k-j_2}{i_2-j_2}\binom{i_2}{j_2}\binom{n-k}{j_2}$$

Ici, nous avons procédé comme précédemment en traitant séparément x, x' d'un côté et y, y' de l'autre côté. Les variables i_1, j_1 étant utilisées sur le chemin de x vers x' et i_2, j_2 sur celui de y à y'. Nous remarquons que nous n'avons pas affiné la limite en considérant qu'il peut y avoir des échanges de héros entre les deux équipes à tout moment. Ceci est fait pour faciliter la preuve. Cependant, cette situation doit être gardée à l'esprit lors de l'introduction des poids de chemin qui ne peuvent pas être égaux à $\frac{1}{(i_1+i_2)!}$ pour cette raison. En fait, au lieu de calculer la valeur exacte des chemins (tout aussi probables), on la borne supérieurement

par $\frac{1}{i_1!i_2!}$.

$$\begin{aligned}
\mathscr{R}(f) \leq & \max_{(e^-,e^+)} \frac{1}{\pi(e^-)P(e^-,e^+)} \sum_{i_1=0}^{k} \sum_{j_1=0}^{i_1-1} \binom{k}{i_1}\binom{n-k-j_1}{i_1-j_1}\binom{i_1}{j_1}\binom{n-k}{j_1}\cdot \\
& \sum_{i_2=0}^{k} \sum_{j_2=0}^{i_2-1} \binom{k}{i_2}\binom{n-k-j_2}{i_2-j_2}\binom{i_2}{j_2}\binom{n-k}{j_2} \frac{1}{i_1!i_2!}\pi_{\max}^2 \\
\leq & \frac{s_{\max}^3}{s_{\min}^3} \frac{1}{\binom{n}{k}\binom{n-k}{k}} 2k(n-k) \sum_{i_1=0}^{k} \sum_{j_1=0}^{i_1-1} \frac{1}{i_1!}\binom{k}{i_1}\binom{n-k-j_1}{i_1-j_1}\binom{i_1}{j_1}\binom{n-k}{j_1}\cdot \\
& \sum_{i_2=0}^{k} \sum_{j_2=0}^{i_2-1} \frac{1}{i_2!}\binom{k}{i_2}\binom{n-k-j_2}{i_2-j_2}\binom{i_2}{j_2}\binom{n-k}{j_2} \\
& \text{car on a maintenant } P(e^-,e^+) \geq \frac{s_{\min}}{s_{\max}}\frac{1}{2k}\frac{1}{n-k} \\
\leq & \frac{s_{\max}^3}{s_{\min}^3} 2k(n-k) A_1 \cdot A_2
\end{aligned}$$

$$\begin{aligned}
\text{avec } A_1 = & \sum_{i_1=0}^{k} \sum_{j_1=0}^{i_1-1} \frac{1}{i_1!}\frac{1}{\binom{n}{k}}\binom{k}{i_1}\binom{n-k-j_1}{i_1-j_1}\binom{i_1}{j_1}\binom{n-k}{j_1} \text{ et} \\
A_2 = & \sum_{i_2=0}^{k} \sum_{j_2=0}^{i_2-1} \frac{1}{i_2!}\frac{1}{\binom{n-k}{k}}\binom{k}{i_2}\binom{n-k-j_2}{i_2-j_2}\binom{i_2}{j_2}\binom{n-k}{j_2}.
\end{aligned}$$

Maintenant, on souhaite avoir une limite supérieure sur A_1 et A_2. Pour A_1, on utilise le même processus que nous avons fait pour l'échantillonnage d'une équipe dans la Section 2.1, ce qui nous donne $A_1 \leq e^4$.

Nous allons maintenant nous concentrer sur A_2. De manière équivalente, nous faisons le développement suivant.

$$\begin{aligned}
A_2 & \leq \sum_{i_2=0}^{k} \sum_{j_2=0}^{i_2-1} \frac{1}{i_2!}\frac{1}{\binom{n-k}{k}}\binom{k}{i_2}\binom{n-k-j_2}{i_2-j_2}\binom{i_2}{j_2}\binom{n-k}{j_2} \\
& \leq \sum_{i_2=0}^{k} \sum_{j_2=0}^{i_2-1} \frac{k!k!}{i_2!(k-i_2)!j_2!(i_2-j_2)!j_2!(i_2-j_2)!} \cdot \frac{(n-2k)!}{(n-k-i_2)!} \\
& = \sum_{i_2=0}^{k} \frac{k\ldots(i_2+1)\cdot k\ldots(k-i_2+1)}{(n-k-i_2)\ldots(n-2k+1)} \sum_{j_2=0}^{i_2-1} \left(\frac{1}{j_2!(i_2-j_2)!}\right)^2 \\
& \leq \sum_{i_2=0}^{k} \frac{k\ldots(i_2+1)\cdot k\ldots(k-i_2+1)}{(n-k-i_2)\ldots(n-2k+1)} \frac{4^{i_2}}{i_2!}
\end{aligned}$$

Avec la même idée, nous nous concentrerons sur deux cas : lorsque $i - i_2 < i_2$ et lorsque $k - i_2 \geq i_2$ en utilisant également l'hypothèse $k^2 \leq n - 2k$. On obtient alors :

$$A_2 \leq e^4$$

ce qui conclu la preuve.

□

2.3 Application sur Dota

Dans la section précédente, nous avons obtenu une majoration grâce à la méthode de flux multiproduits. Ce qui nous donne donc une borne sur le temps de mélange pour l'algorithme d'échantillonnage de Metropolis-Hastings :

$$\tau_x(\epsilon) \leq \frac{s_{\max}^3}{s_{\min}^3} 2k(n-k)e^8 \left(\ln \pi(x)^{-1} + \ln \epsilon^{-1}\right)$$

Pour mettre en œuvre l'algorithme d'échantillonnage, nous devons disposer d'un réseau de neurones qui donne les scores pour chaque match possible. Ce réseau de neurones a été entraîné avec une base de données enregistrant les matchs *Dota 2* à la mi-décembre 2015 [6]. Cette base de données enregistre 3,5 millions matchs avec de nombreuses variables. Pour notre propre usage, nous prenons uniquement les dix identifiants des héros pour chaque match ainsi que le nom de l'équipe gagnante, la première équipe (l'équipe Radiant) ou la seconde (l'équipe Dire).

De plus, nous avons divisé la base de données pour avoir une base de données avec 2 millions de matchs pour entraîner le réseau neuronal, et la deuxième base de données avec 1 million de correspondances pour tester l'entrainement. Le réseau neuronal que nous avons utilisé comporte quatre couches dont deux couches cachées. La couche d'entrée comporte 272 de neurones, qui sont en fait deux vecteurs de taille 136 chacun prenant des valeurs dans $\{0; 1\}$, 1 si le héros correspondant est dans l'équipe et 0 sinon. Les deux couches cachées suivantes allant vers la gauche jusqu'à la couche de sortie ont respectivement 64 et 8 neurones. Pour la mise à jour des paramètres des réseaux de neurones, la méthode utilisée est celle de la descente de gradient. La sortie du réseau neuronal est une couche de taille 1, représentant l'équipe gagnante, rapportant 1 si la première équipe gagne et 0 sinon.

Après le processus d'apprentissage, nous avons environ 61% de prédiction correcte sur la base de données de test. Ce résultat est plutôt bon étant donné que l'on sait que les matchs sont assez bien équilibrés et que généralement la probabilité qu'une équipe gagne est d'environ $0,6$ en général. À titre de test de vérification, remarquant qu'idéalement nous devrions avoir $s(x, y) + s(y, x) = 1$, nous avons calculé cette somme sur 1 million de matchs aléatoires et observé que le score total est d'environ 981098, ce qui est plutôt bien sachant que nous n'avons pas saisi la contrainte selon laquelle il existe une symétrie entre x et y. Ainsi, avec ces résultats, nous pouvons commencer à échantillonner des paires d'équipes.

Comme les scores extremum donnés par le réseau de neurones interviennent sur le taux de convergence, il faut le borner pour éviter d'avoir un temps de mélange trop important. Une façon de procéder comme indiquée précédemment consiste à

modifier la dernière fonction d'activation afin que ses valeurs soient à l'intérieur de l'intervalle $[s_{\min}, s_{\max}]$. Nous avons essayé avec $[0.1, 0.9]$ (au lieu de l'intervalle par défaut $[0, 1]$). Avec cette modification, la limite supérieure du temps de mixage reste raisonnable alors que les performances de l'algorithme sont toujours autour de 59%. Si nous le calculons pour $\epsilon = 0,01$, $k = 5$ et $n = 136$, nous obtenons que le nombre d'étapes est limité par environ 10^{11}, ce qui est un nombre assez grand mais toujours linéaire en $k \cdot n$.

Enfin, comme le taux de convergence correspond au pire des cas, nous avons effectué des simulations pour vérifier si, en pratique, la vitesse de convergence pourrait être meilleure. Ceci est montré sur la Figure 2.2 où l'on remarque que la convergence est atteinte après des pas d'environ 40. L'autocorrélation est calculée en suivant les scores des matchs selon des étapes d'échantillonnage de 10 000. Bien sûr, la véritable vitesse de convergence devrait être comprise entre des pas de 40 et 10^{12} .

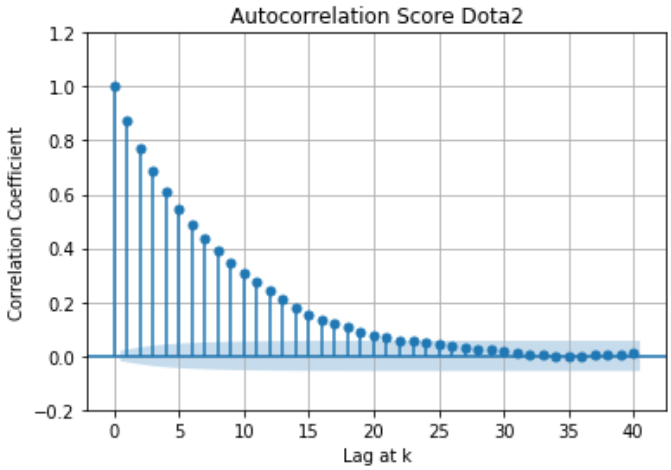

FIGURE 2.2 – Graphique d'autocorrélation avec un échantillonnage par étapes de 10.000.

Pour conclure, dans ce chapitre, nous avons décrit un algorithme pour échantillonner des équipes de cinq héros dans Dota 2. Nous espérons que cette approche aidera les développeurs d'intelligence artificielle pour les jeux vidéo où il pourrait être utile de former l'IA dans les cas les plus difficiles. Cette étude a été fait pour le jeu Dota 2, mais cette approche pourrait être utile pour d'autres jeux du même style.

Chapitre 3

Estimation valeurs de Shapley

3.1 Introduction

3.1.1 Valeur de Shapley

Les valeurs de Shapley permettent d'identifier la contribution de chaque participant lors d'un jeu coopératif. Sachant un gain global, cet outil permet de répartir ce gain aux différents intervenants en fonction de leur participation. Le nom est donnée en l'honneur de Lloyd Shapley qui a introduit cet outil en 1953. Il a notamment remporté le prix Nobel des sciences économiques en 2012.

Plus formellement, on dispose d'un jeu à n joueurs, représenté par l'ensemble $\Omega = \{1, \ldots, n\}$. On pose la fonction caractéristique $\nu : 2^n \to \mathbb{R}$ qui évalue la coalition d'un sous-ensemble de joueurs, avec $\nu(\emptyset) = 0$ où $\emptyset$ désigne l'ensemble vide. Si on pose S une coalition de joueurs, alors $\nu(S)$ décrit la somme totale attendue des gains pour les membres de cette coalition S. En utilisant la valeur de Shapley, le gain donné au joueur i dans un jeu de coalition (ν, Ω) est donné par :

$$\phi_i(\nu) = \frac{1}{n} \sum_{S \subseteq \Omega \setminus i} \binom{n-1}{|S|}^{-1} (\nu(S \bigcup i) - \nu(S)) \tag{3.1}$$

L'expression précédente peut être interprétée comme suit. Pour un joueur i donné, on fait la moyenne des contributions marginales en considérant les poids combinatoires des sous-ensemble de joueurs S. Une autre interprétation possible est mise en évidence quand l'on décompose la somme. Si l'on réécrit la somme en n sommes dont on impose une taille sur S, on remarque que l'objectif est de donner

la même importance pour tous les tailles de sous-ensembles. Explorons maintenant quelques propriétés théoriques sur les valeurs de Shapley.

En regardant en détail l'expression (3.1), on remarque que les valeurs de Shapley possèdent une propriété de symétrie. En effet, si pour tout $S \subseteq \Omega \backslash \{i, j\}$ l'égalité suivante est vérifiée $\nu(S \bigcup i) = \nu(S \bigcup j)$ alors les deux valeurs de Shapley ϕ_i et ϕ_j sont égales. De plus, on peut aussi remarquer que si un joueur ne contribue jamais au score, c'est à dire $\forall S \subset \Omega \backslash i$ on a $\nu(S \bigcup i) = \nu(S)$ alors la valeur de Shapley $\phi_i(\nu) = 0$.

Plusieurs autres expressions équivalentes à (3.1) existent mais c'est à partir de cette expression que nous allons développer une méthode pour trouver une valeur approchée lorsque l'on utilise un réseau de neurones.

3.1.2 Processus ponctuel déterminantal

Un processus ponctuel stochastique est un modèle probabiliste permettant d'étudier un phénomène aléatoire au cours du temps. De plus, le terme ponctuel désigne un type particulier de processus stochastique pour lequel une réalisation est un ensemble de points. Ainsi, un processus ponctuel $\mathcal{P}$ sur un ensemble $\mathcal{Y}$ est une mesure de probabilité sur une "configuration de point" de $\mathcal{Y}$, qui est un sous-ensemble fini de $\mathcal{Y}$. Dans la suite du document, on se concentre seulement sur les processus ponctuel fini discret, et on suppose donc que $\mathcal{Y} = \{1, \ldots, N\}$.

Dans ce cas discret, un processus ponctuel peut être vu comme étant une mesure de probabilité sur $2^{\mathcal{Y}}$, l'ensemble de tous les sous-ensembles de $\mathcal{Y}$. Ainsi, une réalisation provenant de $\mathcal{P}$ peut être l'ensemble vide, l'ensemble $\mathcal{Y}$ tout entier, ou un quelconque sous-ensemble de $\mathcal{Y}$.

Enfin, un processus ponctuel déterminantal est un processus ponctuel stochastique dont la distribution de probabilité est caractérisée par le déterminant d'une fonction. D'une autre manière, $\mathcal{P}$ est appelé processus ponctuel déterminantal si pour un sous-ensemble aléatoire Y provenant d'un tirage de $\mathcal{P}$, on a :

$$\forall A \subset \mathcal{Y}, \mathcal{P}(A \subset Y) = \det(K_A) \tag{3.2}$$

où K est une matrice réelle symétrique $N \times N$ indexé par les éléments de $\mathcal{Y}$. Dans l'expression précédente, on utilise la notation K_A pour $[K_{ij}]_{i,j \in A}$ qui est la restriction de K aux éléments indexé par A. On note alors $\mathcal{P} \sim \mathrm{DPP}(K)$.

Comme $\mathcal{P}$ est une mesure de probabilité, tous les mineurs principaux $\det(K_A)$ de K doivent être positifs ou nuls, ce qui signifie que la matrice K doit être semi-défini positive. On nomme généralement cette matrice comme étant le *noyau* du processus ponctuel déterminantal.

Ce noyau contient l'ensemble des informations que nous avons besoin pour calculer la probabilité qu'un sous-ensemble A soit inclus dans Y. Pour mieux comprendre cela, regardons ensemble quelques cas. Si on suppose que $A = \{i\}$ est un singleton, alors l'expression (3.2) devient :

$$\mathcal{P}(i \in Y) = \det(K_{ii}).$$

Ainsi, les éléments diagonaux de K nous donne les probabilités marginales d'inclusion pour les éléments individuels de $\mathcal{Y}$. Pour des entrées diagonales proches de 1 correspondent aux éléments de $\mathcal{Y}$ souvent sélectionnés par le DPP.

De plus, si on considère maintenant le cas où $A = \{i, j\}$ est un ensemble à deux éléments, alors l'expression (3.2) devient :

$$\begin{aligned} \mathcal{P}(i, j \in Y) =& \begin{vmatrix} K_{ii} & K_{ij} \\ K_{ji} & K_{jj} \end{vmatrix} && (3.3) \\ =& \mathcal{P}(i \in Y)\mathcal{P}(j \in Y) - K_{ij}^2. && (3.4) \end{aligned}$$

Par conséquent, les éléments hors diagonaux déterminent les corrélations négatives entre les paires des éléments. Si la valeur de K_{ij} est élevée alors, i et j ont tendance à ne pas coexister. C'est pour cela que l'on dit que les processus ponctuels déterminantaux permettent une forme de diversification. Si nous considérons les entrées du noyau comme des mesures de similarité entre des paires d'éléments dans $\mathcal{Y}$, alors il est peu probable que les éléments fortement similaires apparaissent ensemble. A l'inverse, lorsque K est diagonale, il n'y a pas de corrélations et les éléments apparaissent indépendamment.

3.1.3 L-ensembles

Pour aider la modélisation sur des données réelles, il est possible de restreindre légèrement la classe des processus ponctuels déterminantaux en nous concentrant sur les L-ensemble. Cette méthode a été introduite par Borodin and Rains dans [3]. Au lieu de définir le DPP par son noyau K, un L-ensemble définit le DPP à travers une matrice réelle symétrique L indexée par les éléments de $\mathcal{Y}$:

$$\mathcal{P}_L(Y = A) \propto det(L_A). \tag{3.5}$$

L'avantage ici est que dans l'équation (3.5) les probabilités sont directement spécifiées pour chaque instance possible de Y. Comme pour le noyau K, L doit être aussi semi-défini positif. De plus, pour tout L semi-défini positif peut définir un L-ensemble. La constante de normalisation peut être donnée en remarquant que $\sum_{A \in \mathcal{Y}} \det(L_A) = \det(L + I)$, où I est la matrice d'identité de taille $N \times N$. Plus généralement, on a le théorème suivant.

Théorème 30. *Pour tout $A \subset \mathcal{Y}$,*

$$\sum_{B \subset A \subset \mathcal{Y}} \det(L_A) = \det(L + I_{\overline{B}})$$

où $I_{\overline{B}}$ est une matrice diagonale avec la valeur 1 si la position diagonale correspondant à un élément de $\overline{B} = \mathcal{Y} \backslash A$, et 0 sinon.

Ainsi, on a

$$\mathcal{P}_L(Y = A) = \frac{\det(L_A)}{\det(L + I)}.$$

Dans la suite, on utilise la notation $\mathcal{P}_L(A)$ pour désigner $\mathcal{P}_L(Y = A)$.

3.2 Processus ponctuel pour valeur de Shapley

Précédemment, nous avons utilisé un réseau de neurones pour aider la simulation d'un échantillon spécifique. Notre but maintenant est d'avoir un outil permettant de mieux comprendre le fonctionnement d'un réseau de neurones. On considère p variables $x_1, \ldots, x_p$ en entrée d'un réseau de neurones. Pour savoir quelles variables ont une influence importante sur la sortie, on utilise les valeurs de Shapley. Pour une variable x_i donnée, sa valeur de Shapley peut être calculée de la manière suivante :

$$\phi_i(\nu) = \frac{1}{p} \sum_{S \subset 1,\ldots,p \setminus \{i\}} \binom{p-1}{|S|}^{-1} [\nu(S \cup \{i\}) - \nu(S)]$$

Cependant, le calcul précédent s'avère long si l'on souhaite une valeur exacte ou une bonne approximation de ϕ_i. En effet, dans notre calcul, la fonction de valeur ν correspond à la fonction de prédiction d'un résultat d'apprentissage du réseau de neurones. Donc pour calculer la somme dans l'expression précédente, il faut à chaque nouvel ensemble S refaire un apprentissage (et accessoirement un autre aussi pour l'ensemble $S \cup \{i\}$). C'est pourquoi le calcul d'un seul ϕ_i peut être très long. Une estimation de cette valeur pourrait nous suffire.

Une autre remarque que l'on peut faire sur le calcul de ϕ_i est que à chaque itération de la somme, on calcul la différence $\nu(S \cup \{i\}) - \nu(S)$. Ainsi, si la variable x_i n'ajoute pas ou très peu d'informations quand on s'associe à un ensemble S donné, alors la différence sera très faible. Sa contribution pour le calcul de ϕ_i ne sera pas très importante. En revanche, avec un autre ensemble S', la variable x_i peut être un bon complément d'information. Dans ce cas, la contribution pour le calcul de ϕ_i sera plus importante.

Donc si on se concentre sur un échantillon d'ensemble où la variable x_i donne une bonne contribution, nous pourrons avoir une idée de la valeur de ϕ_i. Une méthode pour avoir cet échantillon est d'utiliser un processus ponctuel déterminantal. En effet, à partir d'une matrice de similarité entre les différentes variables, on peut construire un processus ponctuel grâce à la méthode du L-ensemble.

On suppose connue L la matrice des similarités pour toutes les variables $x_1, \ldots, x_p$. Depuis cette matrice L, on définit L^i la matrice conditionnelle vérifiant :

$$P_L(Y = A \cup \{i\} | \{i\} \subset Y) = P_{L^i}(A)$$

Avec les résultats présents dans [4], on peut avoir une expression pour L^i qui dépend de la matrice L :

$$L^i = \left(\left[(L + I_{\bar{i}})^{-1} \right]_{\bar{i}} \right)^{-1} - I \tag{3.6}$$

où I est la matrice identité de taille $(p-1) \times (p-1)$ et $I_{\bar{i}}$ est la matrice identité de taille $p \times p$ dont la i-ième valeur est mise à zéro. De plus, si L est semi-défini positif, alors L^i l'est aussi. Ainsi, L^i peut définir un L-ensemble sur l'ensemble des variables x_j avec $j \neq i$.

Sachant cela, on estime la valeur de Shapley ϕ_i en réécrivant l'expression de cette dernière.

$$\phi_i(\nu) = \frac{1}{p} \sum_{S \subset 1,\dots,p \setminus i} \frac{\det(I + L^i)}{\det(L^i_S)} \binom{p-1}{|S|}^{-1} [\nu(S \bigcup \{i\}) - \nu(S)] \frac{\det(L^i_S)}{\det(I + L^i)}$$

De cette manière, nous mettons en évidence la possibilité de simuler un échantillon à partir d'un L-ensemble. Ainsi, on estime ϕ_i par :

$$\hat{\phi}_i(\nu) = \frac{1}{np} \sum_{j=1}^{n} \frac{\det(I + L^i)}{\det L^i_{S_j}} \binom{p-1}{|S_j|}^{-1} [\nu(S_j \bigcup \{i\}) - \nu(S_j)]$$

où $S_1, \dots, S_n$ sont des ensembles de variables qui ont été échantillonnés selon un L-ensemble dont la distribution est définie par :

$$P_{L^i}(S_j) = \frac{\det(L^i_{S_j})}{\det(I + L^i)}.$$

Pour connaître la pertinence de cette démarche, nous allons comparer l'estimateur $\hat{\phi}_i(\nu)$ à un autre estimateur. Ici, on souhaite savoir si l'utilisation de l'échantillonnage par un L-ensemble nous donne plus rapidement les informations de la valeur de Shapley que pour un échantillonnage uniforme sur l'ensemble des sous-ensembles des variables possibles. On note alors $\tilde{\phi}_i$ l'estimateur de ϕ_i défini par :

$$\tilde{\phi}_i(\nu) = \frac{1}{np} \sum_{j=1}^{n} 2^{p-1} \binom{p-1}{|S_j|}^{-1} [\nu(S_j \bigcup \{i\}) - \nu(S_j)]$$

où $S_1, \dots, S_n$ sont des ensembles de variables qui ont été échantillonnés par un loi uniforme, avec $\forall S_j$ on a $\mathbb{P}(Y = S_j) = 1/2^{p-1}$. C'est donc par rapport à cet estimateur que nous allons comparer la pertinence de l'estimateur $\hat{\phi}_i$. Pour cela, nous allons utiliser une base de données sur des matchs du jeu vidéo *League of Legends*.

3.3 Étude sur *League of Legends*

League of Legends est un jeu vidéo d'arène de bataille en équipe, principal concurrent du jeu *Dota 2* que l'on a évoqué dans le Chapitre 2. Ici aussi, les joueurs incarnent un héros et ont pour objectif de détruire la base de leurs adversaires par équipe de cinq. À la fin de chaque partie, un ensemble de variables est donné à tous les participants, pour faire un résumé de la partie. Ainsi, les joueurs peuvent analyser leurs données afin de progresser pour les prochaines parties.

Une base de donnée comprenant les résumés de 180.000 parties datant de 2014 est disponible sur la plateforme web *Kaggle*. Après modification sur les variables et individus, nous utiliserons pour notre étude un tableau de données comportant 38 variables et 150.000 individus. L'une de ces variables correspond à l'information de victoire ou défaite de l'individu, encodé par 0 ou 1. Nous souhaitons prédire cette variable à partir des autres en utilisant un modèle d'amplification de gradient, *gradient boosting*, facilement accessible avec le package *XGBoost* [5]. Avec un tel modèle, nous pourrons utiliser les estimateurs $\hat{\phi}_i$ et $\tilde{\phi}_i$ des valeurs de Shapley, et les comparer.

Nous allons maintenant expliciter la manière que nous utilisons pour calculer les deux estimateurs. Commençons par l'estimateur $\hat{\phi}_i$. Basé sur un processus ponctuel déterminantal, nous souhaitons d'abord avoir une matrice de similarité L^i pour pouvoir échantillonner des sous-ensembles de variables. Pour cela, à partir d'une matrice L des corrélations entre nos 38 variables, nous utilisons l'expression (3.6) pour répondre à notre besoin. Ainsi, il nous est possible d'échantillonner plusieurs sous-ensembles S_j de variables parmi l'ensemble des variables privé de la variable i dont nous voulons connaître l'estimation de la valeur de Shapley. Ensuite, pour chaque sous-ensemble S_j, on réalise l'apprentissage de deux modèles avec *XGBoost* : un avec l'ensemble S_j et l'autre avec l'ensemble $S_j \cup \{i\}$. Pour cela, on sépare notre base de données en une base d'entrainement de 120.000 individus et une base de test de 30.000 individus. Après que les deux modèles ont appris, on calcule la différence pondérée entre les deux prédictions sur la base de test. Et finalement, on répète l'étape d'échantillonnage des S_j et d'apprentissage autant de fois que l'on souhaite. Pour le second estimateur $\tilde{\phi}_i$, on réalise les même étapes mais au lieu d'échantillonner les sous-ensembles S_j avec un L-ensemble, on échantillonne ces sous-ensembles de façon uniforme.

Une variable qui donne le plus d'informations sur le potentiel de victoire est la variable *Deaths*. Cette variable représente le nombre de fois que le joueur est mort durant sa partie. Les graphiques des Figures 3.1 et 3.2 nous donnent la valeur de Shapley en fonction du nombre de mort pour chaque individu dans notre ensemble de tests. On remarque que si le nombre de morts est faible, la valeur de Shapley a

tendance à être plus important, et inversement. Cette estimation provient de nos deux estimateurs $\hat{\phi}_i$ et $\tilde{\phi}_i$, avec $\hat{\phi}_i$ sur le graphe de droite et $\tilde{\phi}_i$ sur le graphe de gauche. Pour chacun, 1.000 sous-ensembles de variables ont été générés. De plus, pour l'estimateur $\hat{\phi}_i$ défini avec un L-ensemble, la matrice de similarité L utilisée est une matrice des corrélations entre chaque variable.

FIGURE 3.1 – Nombre de morts en fonction de la valeur de Shapley estimée avec $\tilde{\phi}_i$ (1000 itérations)

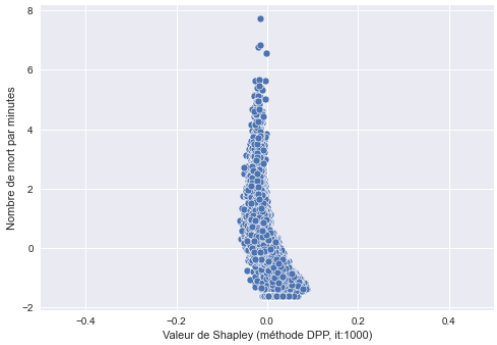

FIGURE 3.2 – Nombre de morts en fonction de la valeur de Shapley estimée avec $\hat{\phi}_i$ (1000 itérations)

On remarque qu'après mille évaluations des différences de prédictions, l'estimateur $\hat{\phi}_i$ nous donne des valeurs moins étalées qu'avec l'estimateur $\tilde{\phi}_i$. Concentrons nous maintenant sur la convergence des deux estimateurs. Pour nous aider, nous allons observer les graphes présents sur les Figure 7 et 6 en annexe. Les deux séries de graphes représentent les estimations des valeurs de Shapley lorsque l'on fait croître le nombre de sous-ensembles échantillonnés (en partant d'un seul sous-ensemble en haut à gauche et terminant à 9 sous-ensembles en bas à droite). On précise que pour l'ensemble des graphes, la fenêtre d'affichage est la même.

Si l'on compare le premier graphe que chaque série, on remarque qu'ils sont tous les deux plus étendu que leur graphe final respectif avec l'estimation sur 1.000 sous-ensembles. De plus, en parcourant les séries, on peut aussi remarquer que pour les estimations avec $\tilde{\phi}$ les graphes commencent à se stabiliser lorsque le nombre de sous-ensembles utilisés est de huit. En revanche, pour les estimations avec $\hat{\phi}_i$, les graphes se stabilise plus tôt, aux alentours de six sous-ensembles. Ainsi, pour ce cas précis, l'estimateur $\hat{\phi}_i$ semble converger plus rapidement. Cependant la différence de temps de convergence n'est pas suffisamment importante pour être significatif dans notre cas.

3.4 Étude sur *Instagram*

Pour notre second étude, nous souhaitons utiliser nos deux estimateurs $\hat{\phi}$ et $\tilde{\phi}$ sur les données *Instagram* pour les comparer empiriquement. *Instagram* est un réseau social de partage de photos et vidéo. Les utilisateurs peuvent partager leur contenu, de façon privé ou public, et peuvent recevoir des appréciations positives de la part des autres utilisateurs, des *'Likes'*. Ce choix a été motivé par une collaboration avec Yunjin Choi de l'Université de Seoul.

Ici, notre base de donnée est composée d'informations liées à 4.000 posts Instagram, répartis entre une quarantaine d'utilisateurs, avec un total de 95 variables. Nous allons utiliser ces différentes informations pour essayer de mieux comprendre comment est-il possible d'avoir un maximum d'appréciations positives. Tout d'abords, concentrons-nous sur la variable "nombre d'Hashtag". Les Hashtag sont des mentions que les utilisateurs peuvent ajouter pour catégoriser leur contenu. Nous commençons par séparer notre base de données en deux : 3.900 individus pour entrainer les modèles et 100 individus comme base de test.

FIGURE 3.3 – Nombre d'Hashtag en fonction de la valeur de Shapley estimée avec $\tilde{\phi}_i$ (1000 itérations)

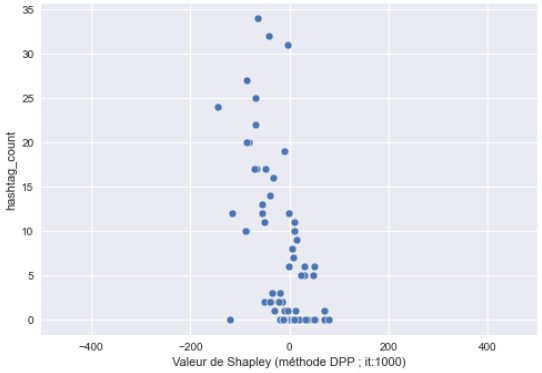

FIGURE 3.4 – Nombre d'Hashtag en fonction de la valeur de Shapley estimée avec $\hat{\phi}_i$ (1000 itérations)

Avec les graphiques des Figures 3.3 et 3.4, on remarque que si un utilisateur ajoute trop de Hashtag sur un post, le nombre de *'Like'* a tendance à être amoindri. Pour avoir ces deux estimations, nous avons procéder de la même manière que pour l'étude sur *League of Legends*, avec 1.000 sous-ensemble de variables. Une nouvelle fois, on remarque que l'estimateur $\hat{\phi}$ donne des valeurs moins étendues que son concurrent $\tilde{\phi}$.

Une autre manière de comparer les deux estimateurs est de regarder leur évolution lorsque l'on augmente progressivement le nombre de sous-ensembles utilisés

pour les estimations. La Figure 3.5 nous donne un aperçu de l'évolution des estimations. L'idée est de comparer les évolutions des estimateurs $\hat{\phi}_i$ et $\tilde{\phi}_i$ pour savoir quel estimateur converge le plus rapidement. Pour cela, pour chaque estimation de ϕ_i avec le nombre de sous-ensembles utilisés n qui varie de 1 à 1.000, on calcule la moyenne des différences absolues entre une estimation et l'estimation avec 1.000 sous-ensembles utilisés. Sur le graphe de gauche avec le courbe bleu, on remarque que l'estimateur $\tilde{\phi}_i$ commence à se stabiliser lorsque l'on utilise plus de 500 sous-ensembles. En revanche, sur le graphe de droite avec la courbe rouge, on remarque l'estimateur $\hat{\phi}_i$ se stabilise plus rapidement, à partir de 200 sous-ensembles utilisés.

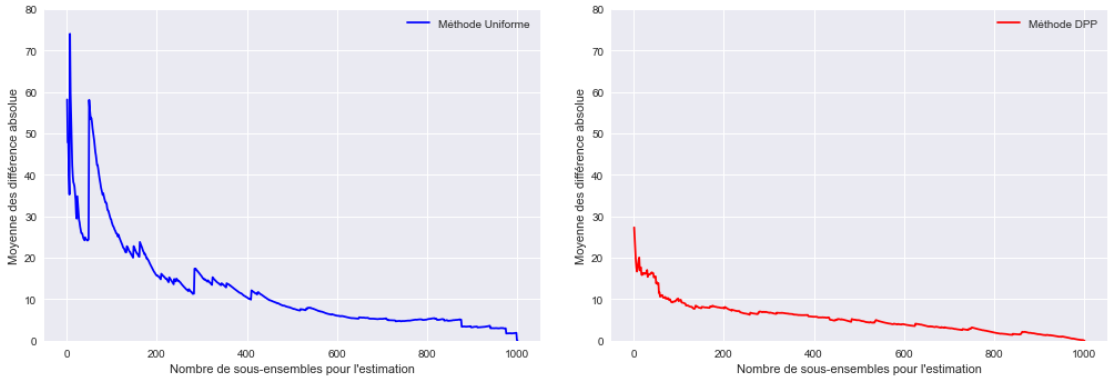

FIGURE 3.5 – Graphes des moyennes de différences absolues entre les estimations par rapport à l'estimation avec 1000 sous-ensembles (à gauche pour $\tilde{\phi}_i$ et à droite pour $\hat{\phi}_i$)

Bibliographie

[1] David Aldous. Random walks on finite groups and rapidly mixing markov chains. In Jacques Azéma and Marc Yor, editors, Séminaire de Probabilités XVII 1981/82, Lecture Notes in Mathematics, pages 243–297, Berlin, Heidelberg, 1983. Springer.

[2] Noga Alon. Eigenvalues and expanders. Combinatorica, 6(2) :83–96, June 1986.

[3] A. Borodin and E. Rains. Eynardmehta theorem, Schur process and their Pfaffian analogs. Journal of statistical physics, 121 :291–317, 2005.

[4] Alexei Borodin and Eric M. Rains. Eynard-Mehta theorem, Schur process, and their pfaffian analogs. Journal of Statistical Physics, 121(3-4) :291–317, November 2005.

[5] Tianqi Chen and Carlos Guestrin. XGBoost : A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 785–794, August 2016.

[6] Albert Cui, Howard Chung, and Nicholas Hanson-Holtry. Yasp 3.5 million data dump.

[7] Nathalie Dereymaeker. Le plan-relief de Lille. Fiabilité et contexte d'une représentation miniature PhD thesis, Université de Lille ; Université catholique de Louvain (1970-....). Faculté de philosophie et lettres, December 2020.

[8] Persi Diaconis and Daniel Stroock. Geometric Bounds for Eigenvalues of Markov Chains. The Annals of Applied Probability, 1(1) :36–61, February 1991. Publisher : Institute of Mathematical Statistics.

[9] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Yoshua Bengio. Generative adversarial networks. In Advances in Neural Information Processing Systems 27, 2014.

[10] W.K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. Biometrika, 57(1) :97–109, 1970.

[11] Iwan Jensen and Anthony J. Guttmann. Statistics of lattice animals (polyominoes) and polygons. Journal of Physics A : Mathematical and General, 33(29) :L257–L263, July 2000.

[12] Mark Jerrum and Alistair Sinclair. Approximating the Permanent. SIAM Journal on Computing, 18(6) :1149–1178, December 1989.

[13] Vijay Konda and John Tsitsiklis. Actor-Critic Algorithms. In Advances in Neural Information Processing Systems, volume 12. MIT Press, 1999.

[14] John Mason. Counting polyominoes of size 50, April 2023.

[15] M. Mihail. Conductance and convergence of Markov chains-a combinatorial treatment of expanders. 30th Annual Symposium on Foundations of Computer Science, pages 526–531, 1989.

[16] Alistair Sinclair. Improved Bounds for Mixing Rates of Markov Chains and Multicommodity Flow. Combinatorics, Probability and Computing, 1(4) :351–370, December 1992.

[17] Alistair Sinclair and Mark Jerrum. Approximate counting, uniform generation and rapidly mixing Markov chains. Information and Computation, 82(1) :93–133, July 1989.

[18] D. Stauffer. Monte Carlo Study of Density Profile, Radius, and Perimeter for Percolation Clusters and Lattice Animals. Physical Review Letters, 41(20) :1333–1336, November 1978.

[19] Solomon W. Golomb. Polyominoes : Pussles, Patterns, Problems, and Packings. Princeton University Press, 2nd edition edition, 1994.

Graphes pour étude sur *League of Legends*

Ici, vous trouverez les graphiques supplémentaires pour l'étude des estimateurs des valeurs de Shapley dans le contexte du jeu *League of Legends*.

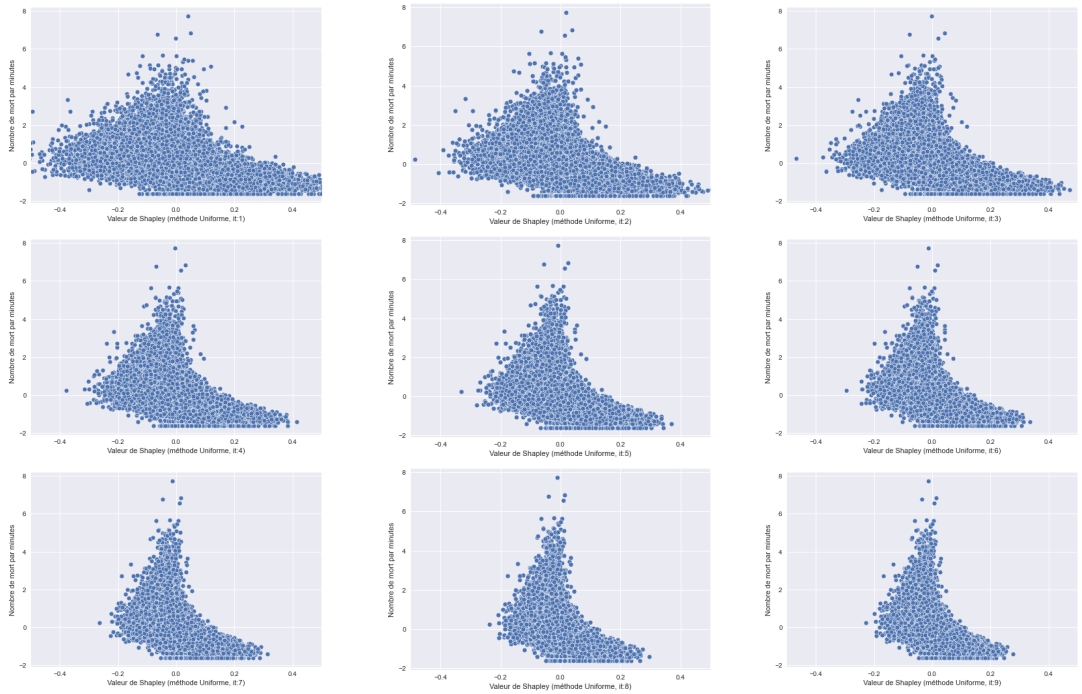

FIGURE 6 – Ensemble des premières valeurs de $\tilde{\phi}$ pour la variable du nombre de morts (avec le nombre de sous-ensembles qui varient de 1 à 9).

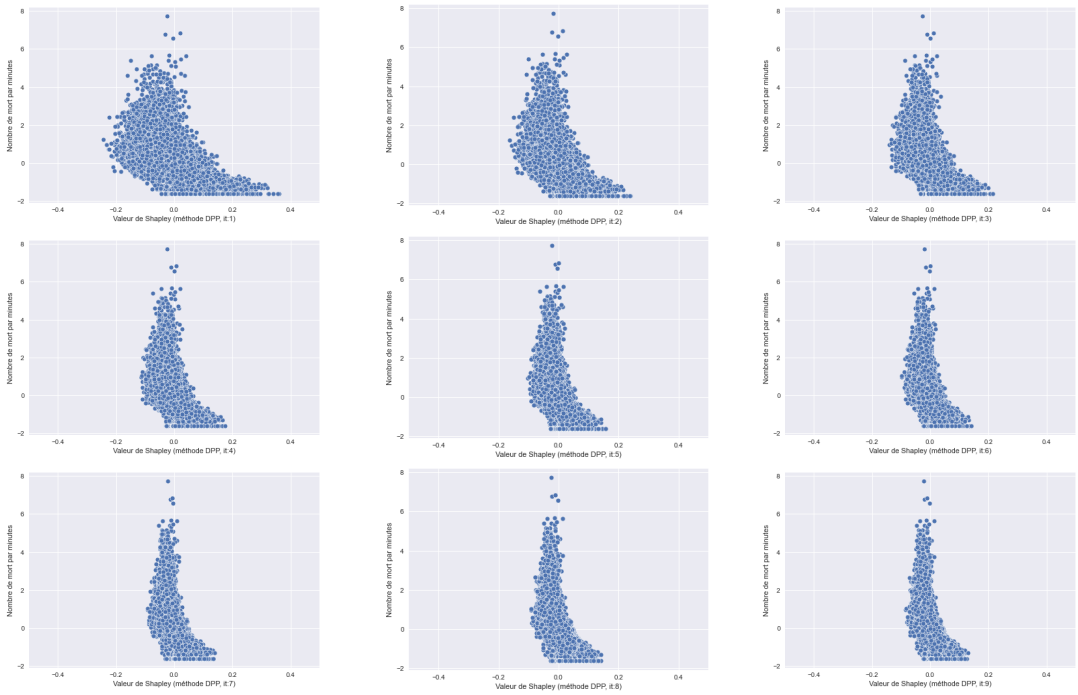

FIGURE 7 – Ensemble des premières valeurs de $\hat{\phi}$ pour la variable du nombre de morts (avec le nombre de sous-ensembles qui varient de 1 à 9).