Faculté
des Sciences

Département des Sciences Informatiques

UMONS
Université de Mons

Étude sur les générateurs de graphes aléatoires

Mémoire réalisé par Nico SALAMONE
pour l'obtention du diplôme de Master en sciences informatiques

Année académique 2018–2019

Directeur: Dr Hadrien MÉLOT

Co-directeur: Dr Karl GROSSE-ERDMANN

Service: Algorithmique

Faculté des Sciences • Université de Mons • Place du Parc 20 • B-7000 Mons

Remerciements

Je tiens à remercier personnellement toutes les personnes qui m'ont aidé à réaliser ce mémoire.

En premier lieu, je remercie mon directeur Dr Hadrien Mélot, chercheur en théorie extrémale des graphes à l'Université de Mons, pour sa disponibilité, son aiguillage à la bonne réalisation de ce travail et sa relecture attentive de ce rapport. De plus, il m'a conseillé un superbe ouvrage m'ayant permis d'utiliser des notations adéquates en matière de graphes.

Je désire également adresser mes remerciements à mon co-directeur Dr Karl Grosse-Erdmann, professeur en probabilités et en statistiques à l'Université de Mons. Il m'a donné de précieux conseils en ce qui concerne les lois de probabilité ainsi que les tests statistiques. Il a en outre réalisé une correction de la section contenant divers rappels sur ces deux domaines.

De même, j'exprime ma gratitude à ma sœur Anastasia Salamone et à son compagnon Maxime Gezels pour avoir relu et corrigé l'orthographe de mon mémoire.

Table des matières

1 Introduction

Les graphes sont grandement utilisés par les chercheurs en science. En effet, ils permettent de modéliser et de résoudre de nombreux problèmes pratiques (problème d'affectation, problème du voyageur de commerce, etc.). Ces chercheurs sont par conséquent fréquemment amenés à générer des graphes aléatoires afin de réaliser des expériences ou des simulations. Plus précisément, ces scientifiques conçoivent des algorithmes impliquant les graphes et testent leur efficacité sur des graphes générés aléatoirement. Dès lors, il est nécessaire que ceux-ci utilisent un *bon* générateur de graphes. Ce type de générateurs est considéré comme bon ou efficace s'il génère aussi souvent un graphe G qu'un autre graphe G'. Prenons un exemple pour illustrer cela. Imaginons qu'un chercheur utilise un générateur de graphes pour tester son algorithme et supposons que ce générateur génère très rarement, voire jamais, un graphe particulier. Cela pourrait être problématique si l'algorithme en question se révèle défaillant sur ce graphe uniquement. Mais comment pouvons-nous alors quantifier l'efficacité d'un générateur ? Comment déterminons-nous qu'un générateur de graphes A est meilleur qu'un autre générateur B ? C'est l'objet de ce mémoire.

Dans notre cas, nous nous attarderons exclusivement aux graphes non orientés et aux générateurs de graphes à n sommets ; n est un de leurs paramètres et ils ne génèrent que des graphes à exactement n sommets. Nous dirons également que deux graphes G et G' sont identiques s'ils sont isomorphes. À l'inverse, nous les déclarerons comme différents si le test d'isomorphisme entre ces deux-ci échoue.

Dans ce rapport, nous ferons dans un premier temps quelques rappels sur la théorie des graphes ainsi que sur les probabilités et les statistiques. Nous présenterons dans un second temps quelques générateurs de graphes aléatoires issus de la littérature ainsi que leur fonctionnement. Ensuite, nous regarderons la distribution des graphes générés par chaque générateur étant donné le nombre de sommets n fixé. Pour chacun, nous aurons donc un diagramme à barres où chacune d'entre elles correspondra à un unique graphe G et dont sa hauteur représentera la fréquence d'apparition de G. Le générateur idéal devrait donc suivre une loi uniforme ; les barres du diagramme seraient au même niveau, signifiant que tous les différents graphes ont la même probabilité d'être générés.

Exemple 1.1. Montrons cela pour les graphes possédant quatre sommets. Il en existe au total 11 uniques ; tous sont illustrés dans la Figure 1.1. Supposons que nous générons 1 100 graphes avec ce générateur idéal. Alors, la distribution de ces graphes sera identique à celle montrée dans le Diagramme 1.1.

Dans la suite de ce rapport, nous élaborerons un évaluateur de générateurs de graphes qui, à partir d'un générateur, calculera un nombre réel représentant la qualité de celui-ci. Il sera ainsi facile de comparer l'efficacité de deux générateurs. Dans un troisième temps, nous analyserons la distribution du nombre d'arêtes de tous les graphes différents à n sommets. Nous en ferons une approximation dans l'intention de créer notre propre générateur de graphes aléatoires. Dans un dernier temps, nous mettrons en œuvre tout ce que nous avons vu durant ce travail dans l'objectif de comparer tous les générateurs étudiés. Pour résumer, ce document sera composé comme suit :

— préliminaires ;

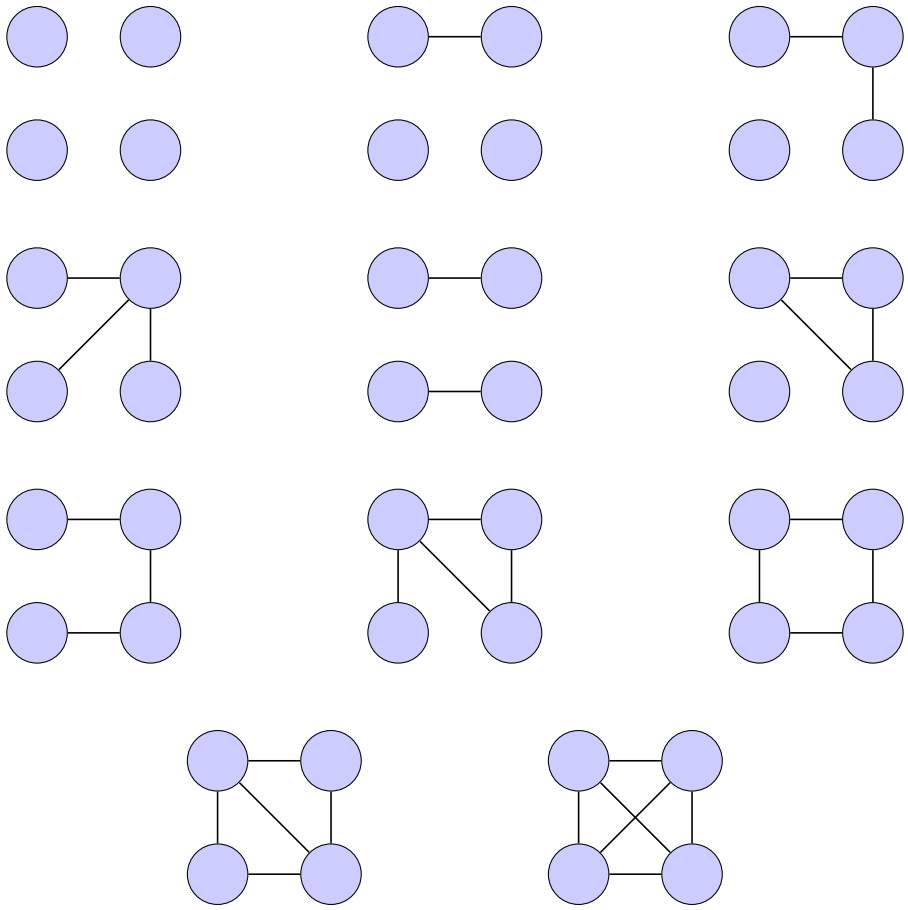


FIGURE 1.1 – Tous les graphes à 4 sommets

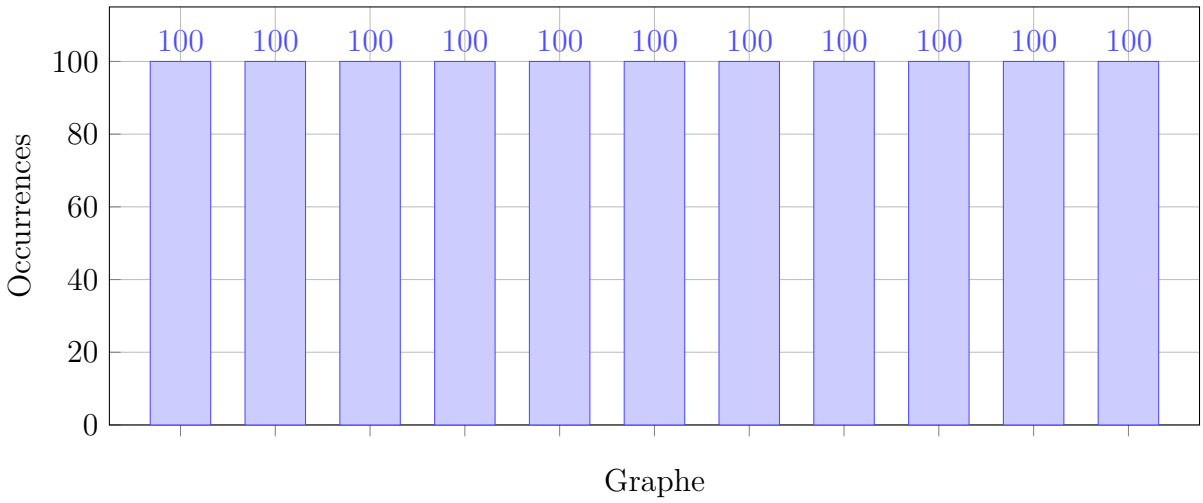


Diagramme 1.1 – Distribution des graphes à 4 sommets générés par le générateur idéal

— évaluation de la qualité d'un générateur ;

— création d'un générateur de graphes ;

— comparaison des générateurs.

Au mieux de notre connaissance, c'est la première étude faite sur la génération de graphes aléatoires à caractère uniforme. Nous avons cependant trouvé d'autres études qui traitaient la génération uniforme de graphes à partir d'une séquence de degrés donnée [13, 17, 18]. En outre, d'autres scientifiques se sont intéressés à l'efficience au niveau du temps de calcul des algorithmes de génération de graphes aléatoires [14,16]. En particulier, les auteurs de l'article [14] proposent un algorithme capable de générer des graphes sur une unité de traitement graphique (GPU).

2 Préliminaires

Dans cette section, nous allons voir quelques notions en théorie des graphes et en probabilités et statistiques nécessaires à la bonne compréhension de ce document. Nous y introduirons également les termes ainsi que les notations associées.

2.1 Théorie des graphes

Graphes

Un *graphe* [3, 4] est une structure de données, disposant celles-ci sous la forme d'un réseau. Les graphes sont très souvent utilisés dans beaucoup de problèmes scientifiques. Leur modélisation rend ceux-ci plus visuels et permet d'en comprendre certaines relations. De surcroît, il existe une quantité impressionnante d'algorithmes s'appliquant sur les graphes et qui ont de nombreuses utilités en pratique.

Définition 2.1. Un *graphe non orienté* G est une paire $G = (V, E)$ dans laquelle V est l'ensemble des *sommets* et E est l'ensemble des *arêtes*.

Une arête est un ensemble de deux sommets u et v. Nous l'écrivons de ce fait $\{u, v\}$. Étant donné qu'une arête est un ensemble, l'ordre des deux sommets n'a pas d'importance et donc l'arête $\{u, v\}$ est identique à l'arête $\{v, u\}$. Observons que la définition ci-dessus n'accepte pas les *arêtes multiples*, c'est-à-dire l'existence de plusieurs arêtes reliant deux sommets u et v.

Définition 2.2. Un *graphe orienté* G est constitué d'une paire $G = (V, E)$ où V est l'ensemble des sommets et E est l'ensemble des *arcs* avec $E \subseteq (V \times V)$.

Malgré la similitude avec l'arête, un arc est une paire de deux sommets (u, v) qui est dirigé de u vers v. L'ordre des deux sommets joue en conséquence un rôle capital ; (u, v) ne représente pas le même arc que (v, u).

Définition 2.3. Soit un graphe $G = (V, E)$ (non orienté ou orienté). L'*ordre* de G est défini comme son nombre de sommets, à savoir $|V|$. Nous le notons $|G|$.

Définition 2.4. Soit un graphe $G = (V, E)$. La *taille* de G, notée $\|G\|$, correspond à son nombre d'arêtes (ou d'arcs), c'est-à-dire $|E|$.

Il est intéressant de noter que la taille d'un graphe G peut être bornée par une valeur dépendante de son ordre $|G|$.

Proposition 2.5. *Soient un graphe orienté $G = (V, E)$ et un graphe non orienté $G' = (V', E')$. Alors, la taille maximale de G est de $|G|^2$ et celle de G' [5] s'élève à :*

$$\binom{|G'|}{2} = \frac{|G'|!}{2!\,(|G'|-2)!} = \frac{|G'|\,(|G'|-1)}{2}.$$

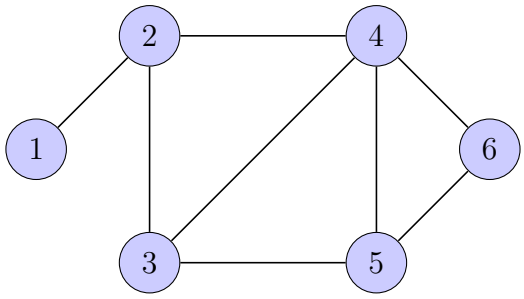


FIGURE 2.1 – Exemple d'un graphe non orienté

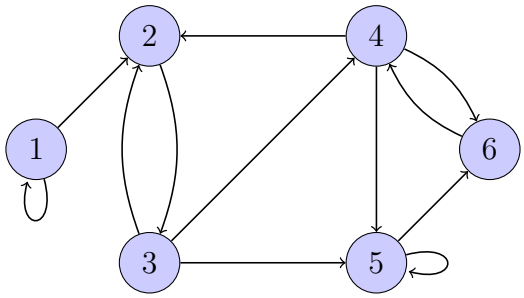


FIGURE 2.2 – Exemple d'un graphe orienté

Tout au long de ce rapport, nous noterons la taille maximale d'un graphe G par $\gamma(|G|)$.

Une *boucle* est un arc dirigé d'un sommet u vers lui-même. Nous l'écrivons donc comme ceci : (u, u). Par définition d'une arête, un graphe non orienté ne peut posséder de boucle.

Un *graphe simple* est un graphe $G = (V, E)$ dont $E \subseteq (V \times V) - \{(u, u) \mid u \in V\}$, c'est-à-dire que G ne possède aucune boucle. Dès lors, tous les graphes non orientés sont des graphes simples.

Exemple 2.6. Le graphe de la Figure 2.1 est un graphe non orienté $G = (V, E)$ (et donc simple) possédant six sommets et huit arcs. Par conséquent, l'ordre de G, $|G|$, est égal à 6 et sa taille, $\|G\|$, est de 8. Nous avons également que $V = \{1, 2, 3, 4, 5, 6\}$ et $E = \{\{1, 2\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{3, 5\}, \{4, 5\}, \{4, 6\}, \{5, 6\}\}$.

Exemple 2.7. La Figure 2.2 montre un graphe orienté $G' = (V', E')$. Son ordre $|G'|$ est de 6 et sa taille $\|G'\|$ est de 12. Ce graphe possédant exactement deux boucles: $(1, 1)$ et $(5, 5)$. De plus, $V' = \{1, 2, 3, 4, 5, 6\}$ et $E' = \{(1, 1), (1, 2), (2, 3), (3, 2), (3, 4), (3, 5), (4, 2), (4, 5), (4, 6), (5, 5), (5, 6), (6, 4)\}$.

Rappelons maintenant les notions de fonction injective, fonction surjective et fonction bijective (ou bijection) [1]. Soit une fonction $f : X \to Y$. Celle-ci est injective si

$$\forall x, x' \in X,\ f(x) = f(x') \implies x = x',$$

est surjective si

$$\forall y \in Y,\ \exists x \in X,\ y = f(x)$$

et est bijective si elle est injective et surjective.

Définition 2.8. Soient deux graphes $G = (V, E)$ et $G' = (V', E')$. Ceux-ci sont dits *isomorphes* s'il existe une bijection $\varphi : V \to V'$ telle que

$$\forall u, v \in V,\ uv \in E \iff \varphi(u)\varphi(v) \in E'.$$

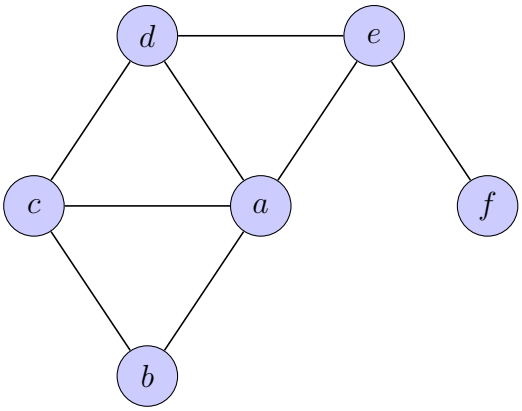


FIGURE 2.3 – Exemple d'un graphe isomorphe à celui de la Figure 2.1

Nous disons que la fonction φ est un *isomorphisme* et notons $G \simeq G'$ la relation d'isomorphisme entre G et G'. En outre, le *problème d'isomorphisme de graphes* consiste à tester que G et G' sont isomorphes.

En réalité, si deux graphes G et G' sont isomorphes, nous avons que G et G' sont identiques à un renommage de sommets près ; il suffit de renommer les sommets de G afin d'obtenir G'. Comme dans ce document nous considérerons que G et G' sont identiques s'ils sont isomorphes, nous noterons en général cette relation $G = G'$.

Exemple 2.9. La Figure 2.3 est un graphe isomorphe à celui de la Figure 2.1. Cela peut être vérifié en prenant la fonction bijective φ suivante:

$$\varphi(u) = \begin{cases} f & \text{si } u = 1, \\ e & \text{si } u = 2, \\ d & \text{si } u = 3, \\ a & \text{si } u = 4, \\ c & \text{si } u = 5, \\ b & \text{si } u = 6. \end{cases}$$

Représentation de graphes

Passons maintenant à la représentation d'un graphe. Il en existe plusieurs ; chacune a ses avantages et ses inconvénients et dépend du cas d'application. Par exemple, certaines représentations sont plus efficaces que d'autres pour modéliser un *graphe dense*, à savoir un graphe disposant d'un grand nombre d'arêtes (ou d'arcs). Dans notre cas, nous allons voir uniquement deux représentations : la matrice d'adjacence et le format graph6.

Commençons par la *matrice d'adjacence*.

Définition 2.10. Soit un graphe $G = (V, E)$, nous définissons sa matrice d'adjacence $A = (a_{ij})$ de taille $|G| \times |G|$ comme suit :

$$(a_{ij}) \coloneqq \begin{cases} 1 & \text{si } v_i v_j \in E, \\ 0 & \text{sinon.} \end{cases}$$

La matrice d'adjacence d'un graphe G est donc une matrice binaire.

Exemple 2.11. Reprenons le graphe G de la Figure 2.1 et le graphe G' de la Figure 2.2. Leur matrice d'adjacence, A et A' respectivement, est la suivante:

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}, \quad A' = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

Dans la matrice A, nous constatons que les valeurs inscrites dans la diagonale sont égales à 0. Cela s'explique simplement par le fait que G est un graphe non orienté et ne possède de ce fait pas de boucle. Par ailleurs, nous remarquons que A est une matrice symétrique (le triangle supérieur et le triangle inférieur sont symétriques). Nous justifions cela par le fait qu'un graphe non orienté contient des arêtes, non orientées. Autrement dit, si un tel graphe possède une arête $\{u, v\}$, alors cela signifie que celui-ci dispose aussi d'une arête $\{v, u\}$. Un graphe non orienté peut ainsi être représenté seulement avec le triangle supérieur (ou inférieur) de la matrice d'adjacence. Une matrice d'adjacence d'un graphe G est peu efficace en matière de mémoire lorsque celui-ci est peu dense ($\|G\| \ll |G|^2$). En effet, nous stockons $|G|^2$ informations alors que seules $\|G\|$ sont nécessaires. D'autres représentations, comme les listes d'adjacence [3], ne stockent que les arêtes (ou arcs) présents dans le graphe. L'avantage des matrices d'ajacence est qu'elles sont facilement interprétables par un humain et permettent de savoir en temps constant si une arête entre un sommet u et un sommet v existe. Le livre [3] fait une comparaison plus détaillée entre ces méthodes de représentation.

La deuxième méthode de représentation de graphes que nous allons utiliser dans ce travail est le *format graph6* [10]. Celui-ci permet de représenter un graphe non orienté en une chaîne de caractères très compacte. Par exemple, le format graph6 du graphe de la Figure 2.1 est `EjKW`. Ce graphe peut donc être stocké en seulement quatre octets. L'Annexe A décrit la procédure pour déterminer le format graph6 d'un graphe.

Il est à noter que deux graphes isomorphes ne possèdent pas obligatoirement la même matrice d'adjacence et le même format graph6.

Nous invitons le lecteur à consulter l'ouvrage [4] s'il souhaite en savoir davantage sur les graphes. Nous conseillons également le livre [3] qui présente diverses structures de données, dont les graphes avec leurs algorithmes sous-jacents.

Dans la suite de ce document, nous ne considérerons plus que les graphes non orientés. Pour des raisons de concision, nous parlerons simplement de graphes, en sous-entendant que ceux-ci sont en réalité des graphes non orientés.

2.2 Probabilités et statistiques

Dans la section précédente, nous avons parlé de théorie des graphes. Étant donné que dans ce rapport, nous allons réaliser plusieurs analyses sur ceux-ci, nous avons besoin

d'outils probabilistes et statistiques. C'est le sujet de cette section. Tout ce qui suit a pour référence le livre [20] et le cours [15].

Probabilités

Les probabilités sont un langage mathématique ayant pour but de traiter l'incertitude et l'aléatoire. L'*espace d'échantillons*, ou *univers*, Ω est l'ensemble de tous les résultats possibles d'une expérience. Nous appelons chaque élément ω de Ω une *réalisation* qui est en fait un résultat possible de l'expérience. Un *événement* A est un sous-ensemble de Ω ($A \subseteq \Omega$).

Soit un univers Ω. Pour chaque événement $A \subseteq \Omega$, nous lui assignons un nombre réel compris entre 0 et 1 et disons qu'il s'agit de la *probabilité* de A, notée $\mathbb{P}(A)$.

Définition 2.12. Soit un univers Ω. Une *loi de probabilité*, également appelée *distribution de probabilité*, $\mathbb{P}$ sur Ω est une fonction $\mathbb{P}\colon A \to \mathbb{R}$, avec $A \subseteq \Omega$, respectant les trois conditions suivantes:

1. $\mathbb{P}(A) \geqslant 0$, pour tout $A \subseteq \Omega$;
2. $\mathbb{P}(\Omega) = 1$;
3. si $A_1, A_2, \ldots$ sont des ensembles disjoints, alors:

$$\mathbb{P}\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} \mathbb{P}(A_i).$$

À partir de ces trois conditions, il s'ensuit ces quelques propriétés:

— $\mathbb{P}(\emptyset) = 0$;
— $A \subseteq B \implies \mathbb{P}(A) \leqslant \mathbb{P}(B)$, $\forall A, B \subseteq \Omega$;
— $0 \leqslant \mathbb{P}(A) \leqslant 1$, $\forall A \subseteq \Omega$;
— $\mathbb{P}(A^{\mathsf{c}}) = 1 - \mathbb{P}(A)$, $\forall A \subseteq \Omega$;
— $P(A \cup B) = \mathbb{P}(A) + \mathbb{P}(B) - \mathbb{P}(A \cap B)$, $\forall A, B \subseteq \Omega$.

En pratique, nous travaillons rarement avec des événements. Effectivement, leur manipulation n'est pas des plus évidentes. Au lieu de cela, nous utilisons des valeurs numériques permettant de résumer l'expérience réalisée. C'est ce que nous appelons une *variable aléatoire*.

Définition 2.13. Soit un espace d'échantillons Ω. Une variable aléatoire est une fonction $X\colon \Omega \to \mathbb{R}$ assignant un nombre réel $X(\omega)$ à chaque réalisation $\omega \in \Omega$.

Soient un univers Ω, une variable aléatoire X sur Ω et un ensemble $R \subseteq \mathbb{R}$. Posons:

$$\mathbb{P}(X \in R) = \mathbb{P}(\{\omega \in \Omega \mid X(\omega) \in R\}),$$

où $\mathbb{P}$ est la loi de probabilité induite sur X et x est une valeur particulière de X, c'est-à-dire $\exists \omega \in \Omega$, $x = X(\omega)$. En particulier, de par cette définition, il en découle:

$$\mathbb{P}(X = x) = \mathbb{P}(X \in \{x\}) = \mathbb{P}(\{\omega \in \Omega \mid X(\omega) = x\}).$$

De plus, précisons que la notation $\mathbb{P}_X(R)$ est une notation équivalente à $\mathbb{P}(X \in R)$.

Définition 2.14. Soit une variable aléatoire X. La *fonction de répartition* de X, $F_X : \mathbb{R} \to [0, 1]$, est définie comme:

$$F_X(x) = \mathbb{P}(X \leqslant x), \forall x \in \mathbb{R}.$$

À partir de cette définition, nous pouvons en déduire:

— F_X est non décroissante: $x_1 < x_2 \implies F_X(x_1) \leqslant F_X(x_2), \forall x_1, x_2 \in \mathbb{R}$;

— $\lim_{x \to -\infty} F_X(x) = 0$ et $\lim_{x \to +\infty} F_X(x) = 1$;

— F_X est continue à droite [1], c'est-à-dire que pour tout x:

$$\lim_{y \to x^+} F_X(y) = F_X(x).$$

Une variable aléatoire X est dite *discrète* si elle prend un nombre dénombrable de valeurs; dans le cas contraire, X est *continue* (voir le livre [1] pour la notion d'ensembles dénombrables et indénombrables).

Définition 2.15. Soit une variable aléatoire X discrète prenant un nombre dénombrable de valeurs dans l'ensemble $\{x_1, x_2, \dots\}$. Nous définissons la *fonction de masse* f_X pour X par:

$$f_X(x) = \mathbb{P}(X = x), \forall x \in \{x_1, x_2, \dots\}.$$

Une fonction de masse f_X satisfait en conséquence:

$$f_X(x) \geqslant 0 \quad \text{et} \quad \sum_i f_X(x_i) = 1,$$

où $x_i \in \{x_1, x_2, \dots\}$.

Définition 2.16. Soit une variable aléatoire X continue. Nous définissons la *densité de probabilité* f_X de X comme une fonction vérifiant ces trois conditions:

1. $f(x) \geqslant 0, \forall x \in \mathbb{R}$;
2. $\displaystyle\int_{-\infty}^{+\infty} f_X(x)\mathrm{d}x = 1$;
3. $\mathbb{P}(a \leqslant X \leqslant b) = \displaystyle\int_a^b f_X(x)\mathrm{d}x$, pour tout $a, b \in \mathbb{R}$ tels que $a \leqslant b$.

De même, nous avons:

$$F_X(x) = \mathbb{P}(X \leqslant x) = \int_{-\infty}^{x} f_X(t)\mathrm{d}t, \forall x \in \mathbb{R}. \tag{2.1}$$

À partir de cette définition, il en résulte que:

$$\begin{aligned}\mathbb{P}(a \leqslant X \leqslant b) &= \mathbb{P}(a < X \leqslant b) \\ &= \mathbb{P}(a \leqslant X < b) \\ &= \mathbb{P}(a < X < b) \\ &= F_X(b) - F_X(a),\end{aligned}$$

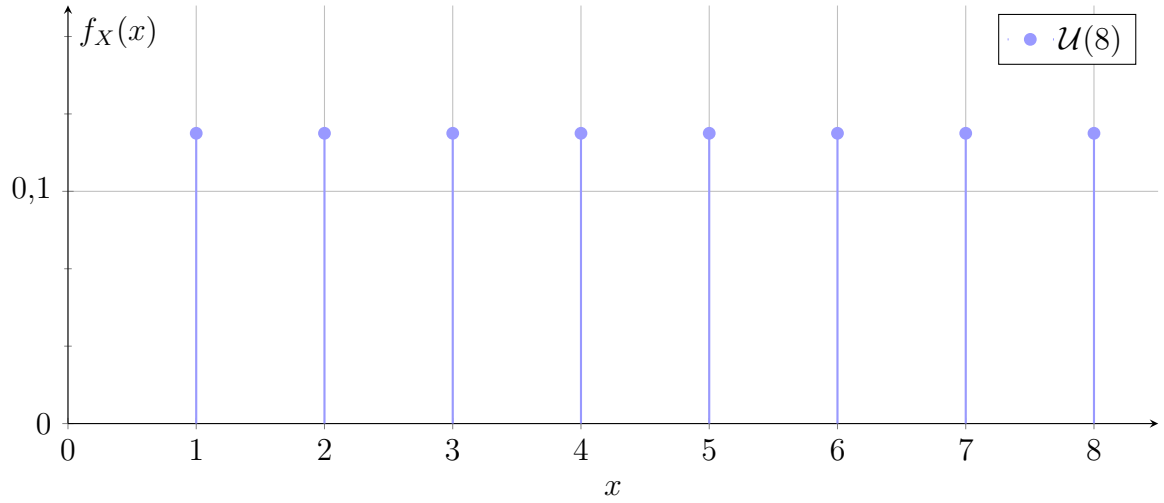


Diagramme 2.1 – Loi uniforme discrète $\mathcal{U}(8)$

pour tout $a, b \in \mathbb{R}$ avec $a \leqslant b$.

Soulignons que la fonction de masse et la densité de probabilité sont deux concepts équivalents. Ces deux fonctions diffèrent simplement par le fait que l'une s'applique sur une variable aléatoire discrète et l'autre sur une variable aléatoire continue. En revanche, l'assertion suivante est fausse pour une densité de probabilité f_X d'une variable aléatoire X:

$$f_X(x) = \mathbb{P}(X = x),\ \forall x \in \mathbb{R}.$$

Nous pouvons expliquer cela par le fait que $\mathbb{P}(X = x) = 0$ pour tout x lorsque X est une variable aléatoire continue.

À présent, présentons diverses lois de probabilité communes. Pour une variable aléatoire X et une loi de probabilité $\mathcal{F}$, nous noterons $X \sim \mathcal{F}$ pour signifier que X suit la loi $\mathcal{F}$.

Définition 2.17. Soient $k \in \mathbb{N}_{>0}$ et une variable aléatoire X discrète. Nous disons que X suit une *loi uniforme discrète* $\mathcal{U}(k)$ si la fonction de masse de X est définie par:

$$f_X(x) = \begin{cases} 1/k & \text{pour tout } x \in \{1, 2, \ldots, k\}, \\ 0 & \text{sinon.} \end{cases}$$

Exemple 2.18. La loi uniforme discrète $\mathcal{U}(8)$ est représentée dans le Diagramme 2.1.

Prenons une expérience avec uniquement deux réalisations possibles: « succès » dans le cas où l'évènement lié se réalise, « échec » sinon. La probabilité de succès est de p et la probabilité d'échec est de $1-p$. Cette expérience est appelée *épreuve de Bernoulli*. Soit X une variable aléatoire discrète correspondant à cette expérience. Nous dirons que X suit une *loi de Bernoulli* de paramètre $p \in [0, 1]$ et noterons cela $X \sim \mathcal{B}(p)$. Nous avons donc que:

$$\mathbb{P}(X = 1) = p \quad \text{et} \quad \mathbb{P}(X = 0) = 1 - p$$

et que la fonction de masse de cette loi de probabilité peut être écrite comme:

$$f_X(x) = p^x(1-p)^{n-x},\ \forall x \in \{0, 1\}.$$

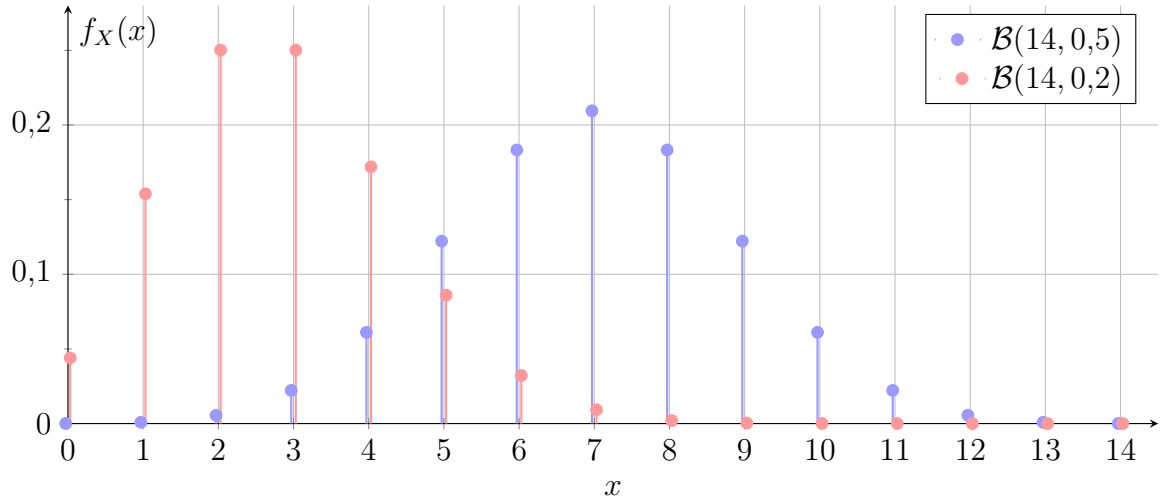


Diagramme 2.2 – Lois binomiales $\mathcal{B}(14, 0{,}5)$ et $\mathcal{B}(14, 0{,}2)$

Définition 2.19. Soient une variable aléatoire X discrète, $r \in \mathbb{N}_{>0}$ et $0 \leqslant p \leqslant 1$. Supposons que nous effectuons r épreuves de Bernoulli de paramètre p de manière indépendantes. Alors, si X représente le nombre de succès de ces r épreuves, X suit dès lors une *loi binomiale* de paramètres r et p, noté $X \sim \mathcal{B}(r, p)$. La fonction de masse f_X de X est par conséquent:

$$f_X(x) = \begin{cases} \binom{r}{x} p^x (1-p)^{r-x} & \text{pour tout } x \in \{0, 1, \ldots, r\}, \\ 0 & \text{sinon.} \end{cases}$$

Exemple 2.20. Le Diagramme 2.2 présente les lois binomiales $\mathcal{B}(14, 0{,}5)$ et $\mathcal{B}(14, 0{,}2)$.

Définition 2.21. Soient une variable aléatoire X continue, $a \in \mathbb{R}$ et $b \in \mathbb{R}$ tels que $a < b$. Cette variable suit une *loi uniforme continue* $\mathcal{U}(a, b)$ si sa densité de probabilité est:

$$f_X(x) = \begin{cases} 1/(b-a) & \text{pour tout } x \in [a, b], \\ 0 & \text{sinon.} \end{cases}$$

Exemple 2.22. Le Diagramme 2.3 contient la loi uniforme $\mathcal{U}(-1, 1)$ ainsi que la loi uniforme $\mathcal{U}(2, 6)$.

Définition 2.23. Soient une variable aléatoire X continue, $\mu \in \mathbb{R}$ et $\sigma > 0$. Cette variable X suit une *loi normale* $\mathcal{N}(\mu, \sigma)$ si la densité de probabilité de X est définie comme suit:

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \forall x \in \mathbb{R}.$$

Le paramètre μ correspond à la moyenne de la densité de probabilité f_X et le paramètre σ est l'écart type de celle-ci. En particulier, lorsque $\mu = 0$ et $\sigma = 1$, nous nommons la loi normale correspondante *loi normale centrée réduite*.

Exemple 2.24. La loi normale centrée réduite $\mathcal{N}(0, 1)$ ainsi que la loi $\mathcal{N}(0, 2)$ sont illustrées dans le Diagramme 2.4.

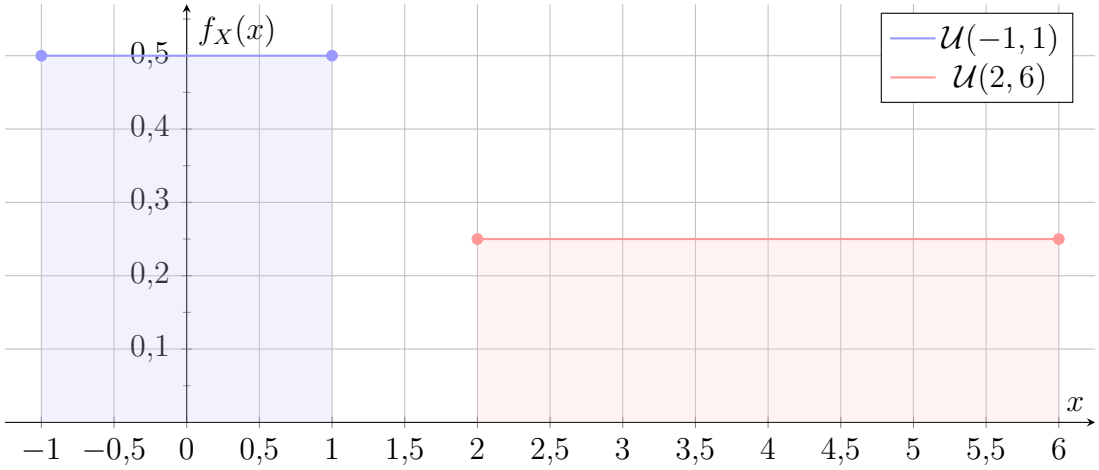


Diagramme 2.3 – Lois uniformes continues $\mathcal{U}(-1,1)$ et $\mathcal{U}(2,6)$

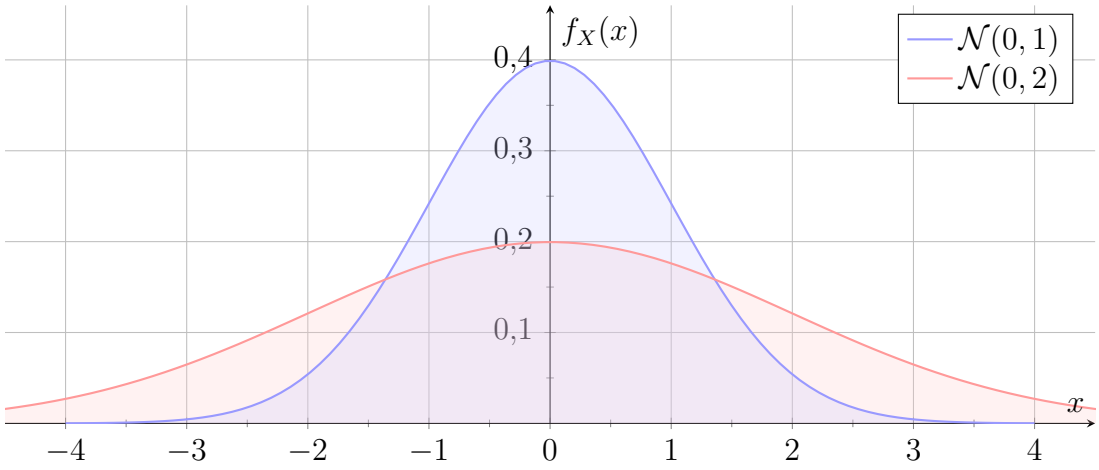


Diagramme 2.4 – Lois normales $\mathcal{N}(0,1)$ et $\mathcal{N}(0,2)$

Il existe un important fait qui lie une loi binomiale à une loi normale. En effet, nous pouvons réaliser une approximation de la loi binomiale $\mathcal{B}(r,p)$ par la loi normale $\mathcal{N}(\mu,\sigma)$ avec

$$\mu = rp \quad \text{et} \quad \sigma = \sqrt{rp(1-p)}.$$

En réalité, la moyenne de la fonction de masse d'une loi binomiale $\mathcal{B}(r,p)$ est de rp et son écart-type est donné par $\sqrt{rp(1-p)}$. Notons toutefois que l'approximation d'une loi normale par une loi binomiale ne peut pas toujours être effectuée.

Tests statistiques

Parlons désormais de *tests statistiques* (ou *tests d'hypothèses*). Chacun de ces tests a pour but de vérifier si une population satisfait une propriété statistique à partir d'échantillons (ou données). Pour ce faire, nous allons utiliser deux hypothèses. L'*hypothèse nulle*, notée H_0, est un fait que nous considérons comme vrai durant l'entièreté du test. D'un autre côté, nous avons l'*hypothèse alternative* H_1 représentant le complémentaire de H_0. L'objectif du test est de tenter de rejeter l'affirmation H_0: soit nous la rejetons au profit de H_1, soit nous ne la rejetons pas. Dans le premier cas, nous pouvons affirmer que H_0 est faux et que H_1 est vrai. Attention cependant, dans le second cas nous ne parvenons pas à rejeter H_0; cela ne signifie pas pour autant que H_0 est vrai et que H_1 est faux. En réalité, nous ne pouvons rien dire: soit H_0 est vrai, soit faux; nous ne savons pas [1].

Comme nous travaillons avec des échantillons aléatoires, cela entraîne une certaine incertitude. De ce fait, il est possible de rejeter H_0 alors que celle-ci est vraie. La probabilité de commettre cette erreur se nomme l'*erreur de première espèce* et est notée α. En pratique, α est souvent fixée à 5% ou 1%. Pourquoi ne pas utiliser une valeur inférieure, proche de 0%, de manière à ce que cette erreur ne se produise que très rarement? La raison est simple: si nous faisons cela, alors notre test statistique ne rejettera jamais H_0 et nous ne pourrons jamais rien conclure.

Pour chaque test statistique, nous calculons la *statistique de test* Z qui est une valeur déterminée à partir des échantillons et de H_0. Également, toujours en supposant que H_0 est vraie, nous pouvons calculer la *p-valeur* (*p-value* en anglais) représentant la probabilité que la statistique de test ait une valeur plus extrême que celle que nous observons dans nos données. Cette valeur nous permet de savoir à quel point le test statistique est significatif; plus il est significatif et plus nous pouvons rejeter H_0 avec une faible probabilité de se tromper. Nous avons donc ceci:

- une p-valeur inférieure à 0,001 indique que le test statistique est extrêmement significatif;
- une p-valeur comprise entre 0,001 et 0,01 nous dit que le test est très significatif;
- une p-valeur comprise entre 0,01 et 0,05 signifie que le test est significatif;
- une p-valeur comprise entre 0,05 et 0,1 représente le fait que le test est assez significatif.

1. Voir le site suivant pour plus d'informations à ce sujet: https://support.minitab.com/fr-fr/minitab/18/help-and-how-to/statistics/basic-statistics/supporting-topics/basics/what-is-a-hypothesis-test/.

De même, nous avons que si la p-valeur est inférieure ou égale à l'erreur de première espèce α, alors nous pouvons rejeter l'hypothèse nulle H_0. Dans le cas contraire, où la p-valeur est strictement supérieure à α, nous ne pouvons pas rejeter H_0.

Parlons du *test d'ajustement du khi-deux*. Celui-ci permet de tester si une variable aléatoire X calculée à partir de n échantillons suit une certaine loi de probabilité $\mathbb{P}$. Ce test est particulièrement efficace pour des données discrètes. Dans le test d'ajustement du khi-deux, l'hypothèse nulle H_0 est le fait que X suit la loi $\mathbb{P}$. L'hypothèse alternative H_1 est le contraire: X ne suit pas la loi. Préalablement, fixons une partition de $\mathbb{R}$ en K classes. Ces dernières sont notées $A_1, A_2, \ldots, A_K$. Pour tout $c \in \{1, 2, \ldots, K\}$, nous calculons l'effectif théorique $T(A_c)$ de la classe A_c comme ceci:

$$T(A_c) = n\mathbb{P}(X \in A_c),$$

ainsi que son effectif observé $O(A_c)$. En fait, la quantité $T(A_c)$ représente le nombre d'échantillons attendus pour la classe A_c et la quantité $O(A_c)$ le nombre d'échantillons observés pour cette même classe A_c. Ensuite, nous calculons la distance d entre T et O:

$$d = \sum_{c=1}^{K} \frac{(O(A_c) - T(A_c))^2}{T(A_c)}.$$

Posons $\hat{p}_c = O(A_c)/n$, cette distance peut être réécrite comme:

$$\sum_{c=1}^{K} \frac{(O(A_c) - n\mathbb{P}(X \in A_c))^2}{n\mathbb{P}(X \in A_c)} = \sum_{c=1}^{K} \frac{(n\,(\hat{p}_c - \mathbb{P}(X \in A_c)))^2}{n\mathbb{P}(X \in A_c)} = n \sum_{c=1}^{K} \frac{(\hat{p}_c - \mathbb{P}(X \in A_c))^2}{\mathbb{P}(X \in A_c)}.$$

La valeur d représente la statistique de test de ce test. Nous pouvons déterminer la p-valeur comme suit:

$$p = \mathbb{P}(\chi^2_{K-1} \geqslant d),$$

où χ^2_{K-1} est une variable aléatoire de la loi du khi-deux à $K-1$ degrés de liberté. Pour plus d'informations sur cette loi, le livre [19] répertorie un grand nombre de lois de probabilité, dont celle-ci. Il est à noter que le test d'ajustement du khi-deux peut s'avérer invalide s'il existe un effectif théorique inférieur à 5. En conséquence, $T(A_c)$ doit être supérieur ou égal à 5 pour toute classe A_c.

Exemple 2.25. Un nouveau jeu de tickets à gratter vient de faire son apparition sur le marché. L'entreprise qui fabrique ces tickets déclare:

> « Ce jeu va complètement révolutionner notre société. Dites fin à la classe populaire et dites bonjour aux îles de Bahamas! »

Voici les gains ainsi que les probabilités respectives annoncées par l'entreprise:

$$\mathbb{P}(X = x) = \begin{cases} 0{,}4 & \text{si } x = 0, \\ 0{,}3 & \text{si } x = 1, \\ 0{,}16 & \text{si } x = 2, \\ 0{,}1 & \text{si } x = 5, \\ 0{,}04 & \text{si } x = 10, \end{cases}$$

Gain (en euro)	**Nombre de tickets avec ce gain**
0	234
1	145
2	78
5	37
10	6

Tableau 2.1 – Résumé des gains obtenus pour les 500 tickets

Classe	**effectif observé**	**effectif théorique**
0	234	$500 \times 0{,}4 = 200$
1	145	$500 \times 0{,}3 = 150$
2	78	$500 \times 0{,}16 = 80$
5	37	$500 \times 0{,}1 = 50$
10	6	$500 \times 0{,}04 = 20$

Tableau 2.2 – Résumé des gains obtenus pour les 500 tickets

où x est le gain en euro.

Nous souhaiterions vérifier ces informations. Pour ce faire, nous avons acheté 500 de ces tickets à gratter et avons compté tous les gains obtenus. Ces données sont reprises dans le Tableau 2.1. Dans cet exemple, l'hypothèse nulle est le fait que celles-ci coïncident avec les données livrées par l'entreprise; en d'autres termes, l'entreprise dit la vérité.

Les classes peuvent raisonnablement représenter les gains; nous en avons donc cinq. Dès lors, chaque donnée présentée dans le Tableau 2.1 correspond à l'effectif observé $O(A_c)$ de chaque classe A_c. Quant à l'effectif théorique $T(A_c)$ de cette dernière, il est déterminé comme suit:

$$T(A_c) = n\mathbb{P}(X \in A_c),$$

où $n = 500$ et $\mathbb{P}(X \in A_c)$ peut être trouvée dans les valeurs fournies par l'entreprise. Le Tableau 2.2 comprend l'ensemble des effectifs observés et des effectifs théoriques calculés.

Calculons à présent la statistique de test d du test d'ajustement du khi-deux:

$$d = \frac{(234-200)^2}{200} + \frac{(145-150)^2}{150} + \frac{(78-80)^2}{80} + \frac{(37-50)^2}{50} + \frac{(6-20)^2}{20} \approx 19{,}177.$$

En ce qui concerne la p-valeur, nous avons:

$$p \approx 0{,}0007.$$

Le test est de ce fait extrêmement significatif. L'hypothèse nulle peut donc être rejetée avec une très faible probabilité de se tromper. En conclusion, nous pouvons dire avec une grande certitude que les probabilités révélées par l'entreprise sont fausses.

Régression linéaire

Nous allons à présent parler de *régression linéaire simple*. Soient X et Y deux ensembles de nombres réels de taille n de la forme:

$$X = \{x_1, x_2, \ldots, x_n\},$$
$$Y = \{y_1, y_2, \ldots, y_n\}.$$

L'objectif est de trouver une fonction mathématique $r(x)$ permettant de prédire *le plus efficacement possible* y_i en fonction de x_i pour tout $x_i \in X$ et $y_i \in Y$. Cette fonction est une droite et possède par conséquent cette forme:

$$r(x) = \beta_0 + \beta_1 x,$$

où β_0 est l'ordonnée à l'origine de la droite et β_1 est la pente de celle-ci. Pour trouver cette droite, nous devons estimer les deux paramètres β_0 et β_1 à partir des données X et Y. Notons les estimations de ceux-ci par $\hat{\beta}_0$ et $\hat{\beta}_1$ respectivement. Il existe plusieurs méthodes d'estimation. Néanmoins, nous n'allons en présenter qu'une seule: la méthode des moindres carrés. Celle-ci nous dit que:

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}, \tag{2.2}$$
$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}, \tag{2.3}$$

où $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ et $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$.

Exemple 2.26. Le Diagramme 2.5 propose un exemple de régression linéaire simple dans lequel $\hat{\beta}_0 = 5{,}5$ et $\hat{\beta}_1 = 2{,}6$ calculés à partir de deux ensembles X et Y grâce aux Équations (2.2) et (2.3).

Dans l'éventualité où le lecteur désire approfondir le sujet, le livre [20] contient de nombreuses informations sur la thématique des probabilités et des statistiques. Notamment, ce livre présente d'autres lois de probabilités et d'autres méthodes d'inférence statistique. Pour aller encore plus loin, le livre [19] propose une quarantaine de lois de probabilité différentes.

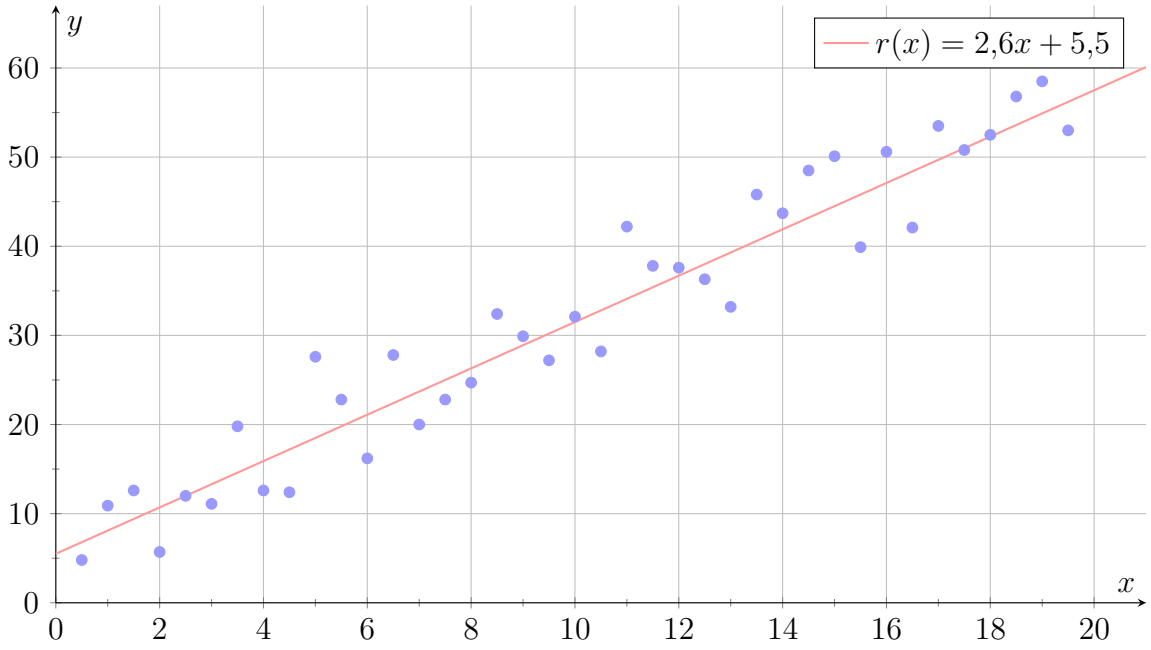


Diagramme 2.5 – Exemple de régression linéaire simple

3 Évaluation de la qualité d'un générateur de graphes aléatoires

Maintenant que nous avons fait quelques rappels sur la théorie des graphes et sur les probabilités et les statistiques, nous allons passer au vif du sujet. Nous allons proposer dans cette section un évaluateur de générateurs de graphes aléatoires. À partir d'un générateur A donné, celui-ci sera capable de quantifier sa qualité de génération; plus la distribution des graphes générés par A sera proche d'une distribution uniforme et plus le générateur sera considéré comme meilleur. Avant de faire cela, nous présenterons deux générateurs de graphes aléatoires bien connus au sein de la littérature et montrerons une façon de résoudre le problème d'isomorphisme de graphes. Grâce à cela, nous pourrons calculer la distribution des graphes générés pour les deux générateurs et pourrons ainsi créer notre évaluateur de générateurs de graphes.

3.1 Générateurs de graphes aléatoires étudiés

Dans la section actuelle, nous allons présenter deux générateurs de graphes: les deux modèles d'Erdős–Rényi.

Le premier modèle, appelé *modèle* $G(n, m)$, a été introduit par Paul Erdős et Alfred Rényi en 1959 [5] et a été davantage étudié en 1960 [6]. Ce modèle permet de générer un graphe aléatoire avec n sommets et m arêtes, où n et m sont les deux paramètres de ce modèle. Soient $n \geqslant 1$ l'ordre du graphe désiré et $m \geqslant 0$ la taille de celui-ci. Disons que $\gamma(n)$ correspond au nombre maximal d'arêtes d'un tel graphe, c'est-à-dire que $0 \leqslant m \leqslant \gamma(n)$. Nous avons vu dans la Section 2.1 que $\gamma(n) = \binom{n}{2}$. Notons $\mathscr{G}_{n,m}$ l'ensemble des graphes d'ordre n et de taille m, à savoir tous les graphes $G = (V, E)$ tels que $|V| = n$ et $|E| = m$ [1]. Cet ensemble contient alors exactement $\binom{\gamma(n)}{m}$ éléments. En fait, le modèle $G(n, m)$ sélectionne aléatoirement (et de façon uniforme) un graphe G parmi tous les graphes dans $\mathscr{G}_{n,m}$. Concrètement, nous prenons un graphe à n sommets et sans aucune arête et lui ajoutons m arêtes choisies aléatoirement dans les $\gamma(n)$ possibles. Ce modèle peut être implémenté de différentes manières. L'Algorithme 3.1 est une implémentation similaire à celle faite dans NetworkX [2], une bibliothèque logicielle Python pour les graphes.

Étant donné que notre objectif est d'avoir un générateur de graphes le plus uniforme possible, nous ne pouvons fixer m, la taille du graphe, à une valeur bien précise. Effectivement, supposons que nous fixons m à 0. Cela signifie que le modèle $G(n, m)$ nous renverrait toujours le même graphe G (celui à n sommets et zéro arête). En conséquence, ce générateur ne pourra pas être uniforme. En réalité, nous avons besoin de faire varier m pour que le générateur soit capable de générer tous les graphes existants à n sommets. Nous introduisons donc l'Algorithme 3.2 qui est un dérivé du modèle $G(n, m)$: il choisit aléatoirement un nombre m entre 0 et $\gamma(n)$ et utilise le modèle $G(n, m)$ avec n et m comme paramètres pour générer le graphe. Nous appelons ce générateur GNM-Der.

Passons au deuxième modèle: le *modèle* $G(n, p)$ [9]. Il a été introduit la même année que le modèle $G(n, m)$, c'est-à-dire en 1959, par Edgar Gilbert. Le modèle $G(n, p)$ génère un

1. Il se peut que certains graphes dans $\mathscr{G}_{n,m}$ soient isomorphes.
2. Voir le site suivant pour consulter la documentation de NetworkX: https://networkx.github.io/documentation/stable/.

Algorithme 3.1 GNM(n, m)

Entrée:
L'ordre n et la taille m du graphe à générer.
Sortie:
Un graphe pris aléatoirement dans $\mathscr{G}_{n,m}$.

1: $G \leftarrow$ Créer un graphe à n sommets et 0 arête
2: $V \leftarrow$ Récupérer les sommets de G
3: $c \leftarrow 0$
4: **tant que** $c < m$ **faire**
5: $u \leftarrow$ Choisir aléatoirement un élément dans V
6: $v \leftarrow$ Choisir aléatoirement un élément dans V
7: **si** $u \neq v$ et G ne possède pas l'arête $\{u, v\}$ **alors**
8: Ajouter l'arête $\{u, v\}$ à G
9: $c \leftarrow c + 1$
10: **retourner** G

Algorithme 3.2 GNM-Der(n)

Entrée:
L'ordre n du graphe à générer.
Sortie:
Un graphe aléatoire à n sommets.

1: $M \leftarrow n(n-1)/2$ # Rappel: $\binom{n}{2} = \frac{n(n-1)}{2}$.
2: $m \leftarrow$ Tirer un nombre aléatoire entre 0 et M uniformément
3: **retourner** GNM(n, m)

Algorithme 3.3 GNP(n, p)

Entrée:
L'ordre n du graphe à générer et la probabilité p de sélectionner chaque arête possible.
Sortie:
Un graphe sélectionné aléatoirement dans $\mathscr{G}_n$.

```
1: G ← Créer un graphe à n sommets et 0 arête
2: V ← Récupérer les sommets de G
3: pour chaque u dans V faire
4:     pour chaque v dans V faire
5:         si u ≠ v alors
6:             r ← Tirer un nombre aléatoire entre 0 et 1 de façon uniforme
7:             si r < p alors
8:                 Ajouter l'arête {u, v} à G
9: retourner G
```

Algorithme 3.4 GNP-Der(n)

Entrée:
L'ordre n du graphe à générer.
Sortie:
Un graphe aléatoire à n sommets.

```
1: retourner GNP(n, 0,5)
```

graphe à n sommets. Le nombre d'arêtes de ce graphe dépend d'un paramètre p; chaque arête possible a une probabilité p d'être sélectionnée. De ce fait, plus p est élevée et plus le graphe généré a de chance d'être dense. Soient $n \geqslant 1$ le nombre de sommets souhaité et $p \in [0, 1]$. L'ensemble de tous les graphes d'ordre n, noté $\mathscr{G}_n$, a une taille de $2^{\binom{n}{2}}$. Le modèle $G(n, p)$ sélectionne un de ces graphes en opérant comme suit: à partir d'un graphe à n sommets ne possédant aucune arête, lui ajouter, avec une probabilité p, chaque arête possible. Il est à noter que l'éventuelle sélection d'une arête $\{u, v\}$ est une variable aléatoire X suivant la loi de Bernoulli $\mathcal{B}(p)$. Une implémentation du modèle $G(n, p)$ est disponible dans l'Algorithme 3.3.

Comme pour le modèle précédent, nous souhaitons que le modèle $G(n, p)$ génère des graphes de la façon la plus uniforme possible. Pour ce faire, nous devons fixer p à 0,5 puisque cette valeur permet de choisir uniformément un graphe dans $\mathscr{G}_n$. Nous dérivons donc le modèle $G(n, p)$ en un nouveau générateur de graphes montré dans l'Algorithme 3.4. Nous nommons ce dernier GNP-Der.

Exemple 3.1. Illustrons le fonctionnement de l'Algorithme 3.1 en supposant que $n = 4$ et $m = 3$. Les différentes étapes sont exposées dans la Figure 3.1. Initialement, nous avons un graphe G à quatre sommets et avec aucune arête. À la première itération, l'arête $\{1, 3\}$ est choisie ($u = 1$ et $v = 3$). Comme celle-ci n'a pas encore été sélectionnée, nous pouvons l'ajouter à G. Lors de la deuxième itération, il se passe la même chose, mais cette fois-ci pour l'arête $\{4, 1\}$. À la troisième itération, l'arrête $\{3, 1\}$ est choisie. Elle n'est pas

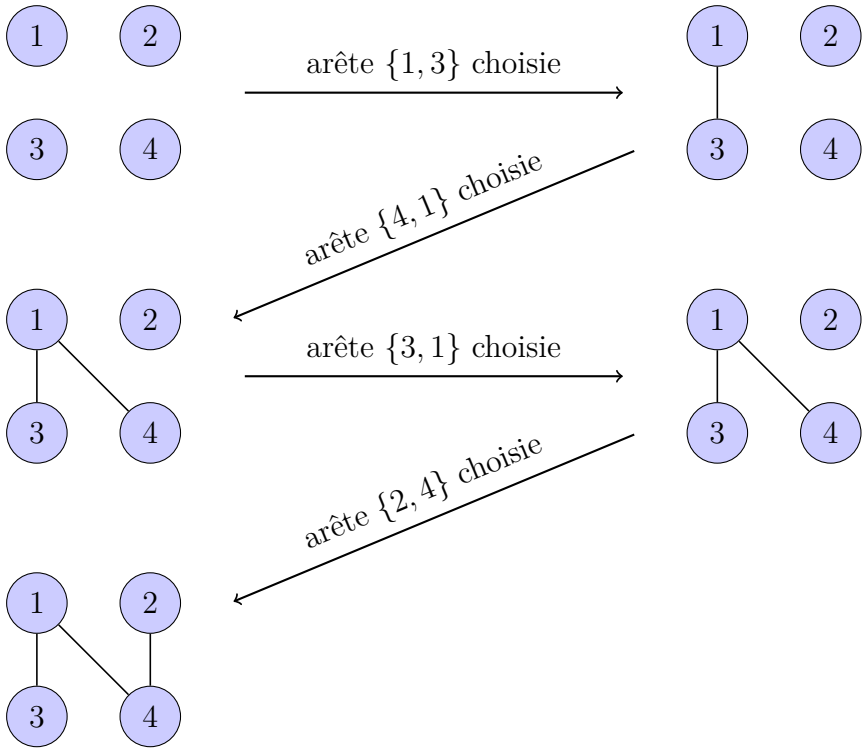


FIGURE 3.1 – Exemple de création d'un graphe avec GNM

ajoutée au graphe G, car l'arête $\{1,3\}$ a déjà été sélectionnée auparavant (rappelons que deux arêtes $\{u,v\}$ et $\{v,u\}$ sont strictement identiques pour tous sommets u et v). Durant la quatrième itération, l'arête $\{2,4\}$ est désignée et insérée dans G vu qu'elle n'est pas encore présente dans celui-ci. Étant donné que trois arêtes ont été sélectionnées ($m = 3$), l'algorithme se termine à ce moment et retourne le graphe G ainsi formé.

Exemple 3.2. Montrons maintenant un exemple de création d'un graphe en utilisant l'Algorithme 3.3. Disons que $n = 4$ et $p = 0{,}4$. La Figure 3.2 recense les principales étapes de l'algorithme. Nous construisons d'abord un graphe G à quatre sommets et sans arête. Toutes les arêtes possibles pour un graphe comprenant quatre sommets sont les suivantes: $\{1,2\}$, $\{1,3\}$, $\{1,4\}$, $\{2,3\}$, $\{2,4\}$ et $\{3,4\}$. Pour chacune, un nombre compris entre 0 et 1 est généré suivant une loi uniforme. Présumons que les nombres qui suivent ont été tirés: 0,208 pour l'arête $\{1,2\}$, 0,101 pour $\{1,3\}$, 0,929 pour $\{1,4\}$, 0,703 pour $\{2,3\}$, 0,092 pour $\{2,4\}$ et 0,131 pour $\{3,4\}$. Puisque 0,208, 0,101, 0,092 et 0,131 sont inférieurs à 0,4 (paramètre p), les arêtes $\{1,2\}$, $\{1,3\}$, $\{2,4\}$ et $\{3,4\}$ sont ajoutées à G. La construction du graphe G est à présent terminée; il est donc retourné.

3.2 Résolution du problème d'isomorphisme de graphes

Nous avons vu dans la section précédente des générateurs de graphes aléatoires. Pour nous assurer que ceux-ci génèrent aussi souvent un graphe G qu'un autre graphe G' en tenant compte de la contrainte d'isomorphisme, nous avons besoin de résoudre le problème d'isomorphisme de graphes. Soient G et G' deux graphes, ce problème peut être résolu de

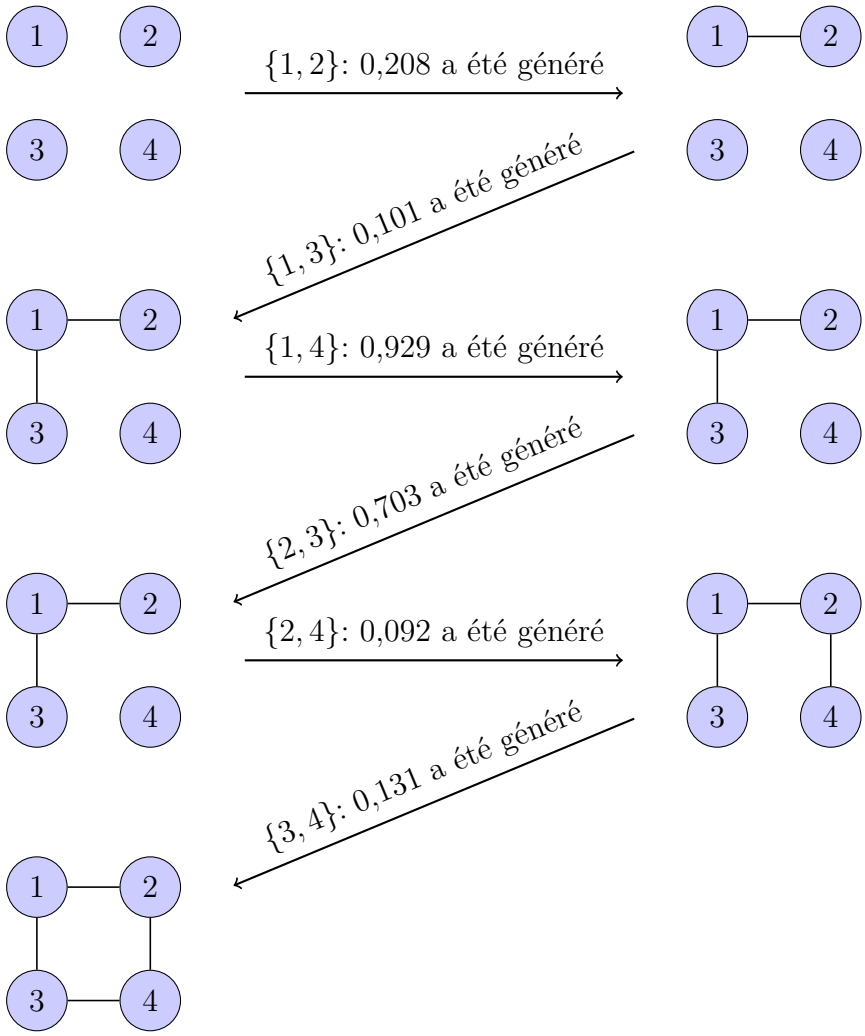


FIGURE 3.2 – Exemple de création d'un graphe avec GNP

deux façons [8]:

1. tester que G et G' sont isomorphes;
2. déterminer la représentation canonique de G et la représentation canonique de G' et comparer ces deux représentations.

Nous allons nous intéresser uniquement à cette deuxième méthode vu qu'elle est plus efficiente en ce qui concerne le temps de calcul que la première [2].

Définition 3.3. Soit un graphe G. Un *algorithme de labellisation canonique* est une fonction C qui calcule la *représentation canonique* $C(G)$ de G [2, 8] telle que G et G' sont isomorphes si et seulement si $C(G) = C(G')$ pour tout autre graphe G'. Formulons cela d'une autre manière. Reprenons $\mathscr{G}_n$, l'ensemble de tous les graphes à n sommets et présumons que, pour chaque $G'' = (V'', E'')$ dans $\mathscr{G}_n$, $V'' = \{1, 2, \ldots, n\}$. Disons aussi que $G \in \mathscr{G}_n$ et $G = (V, E)$. Alors, un algorithme de labellisation canonique assigne une permutation π_G de V de telle sorte que G et tout autre graphe G' dans $\mathscr{G}_n$ sont isomorphes si et seulement si $\pi_G(G)$ correspond à $\pi_{G'}(G')$.

En réalité, un algorithme de labellisation canonique réalise un renommage de sommets de tel façon que deux graphes sont isomorphes si et seulement si les deux graphes après renommage sont identiques.

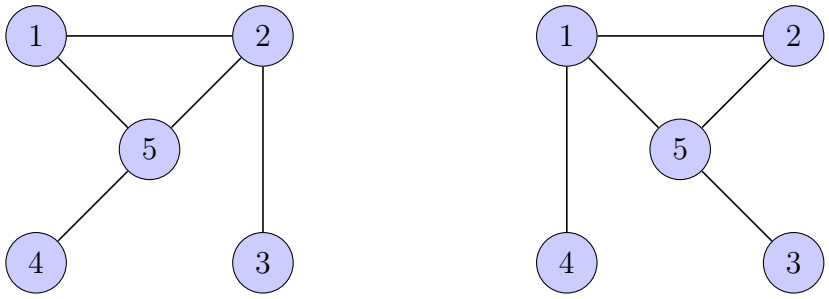
Dans ce travail, nous allons utiliser l'algorithme de labellisation canonique de Brendan Damien McKay. Le programme associé à cet algorithme se nomme **nauty**. Il a été initialement présenté en 1981 dans l'article [11] et a été mis à jour et détaillé dans l'article [12] en 2013. De plus, les grandes étapes de cet algorithme sont décrites dans la référence [8]. En pratique, le programme **nauty** reçoit en entrée la matrice d'adjacence d'un ou plusieurs graphes et retourne la représentation canonique de ceux-ci au format graph6 discuté dans la Section 2.1 et l'Annexe A.

Exemple 3.4. Considérons deux graphes G et G' isomorphes, tous les deux montrés dans la Figure 3.3. Voici leur matrice d'adjacence A et A' respectivement:

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix} \quad \text{et} \quad A' = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{pmatrix}.$$

Quant à leur format graph6, nous avons `Dgs` pour G et `Dcw` pour G'. Après avoir envoyé séparément les deux matrices A et A' à l'algorithme de labellisation canonique de McKay, nous obtenons la même représentation canonique, `DD[`, comme G et G' sont isomorphes. Celle-ci correspond à un nouveau graphe G'', représenté dans la Figure 3.4. La matrice d'adjacence A'' de G'' est la suivante:

$$A'' = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}.$$



(a) Graphe G (b) Graphe G'

FIGURE 3.3 – Deux graphes isomorphes à cinq sommets

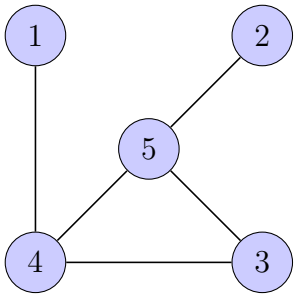


FIGURE 3.4 – Graphe obtenu après avoir appliqué la labellisation canonique

3.3 Distribution des graphes générés par générateur

Dans les deux sections précédentes, nous avons présenté plusieurs générateurs de graphes aléatoires ainsi qu'une méthode permettant de résoudre le problème d'isomorphisme de graphes. Pour chaque générateur A, nous allons dans cette section calculer la distribution des graphes générés par A en utilisant le fait que deux graphes sont identiques si et seulement s'ils sont isomorphes. Cette distribution peut être vue comme un diagramme à barres: chaque barre désignera la représentation canonique de graphes isomorphes et la hauteur de celle-ci désignera le taux d'apparition de ces graphes. De manière plus formelle, pour tous graphes G et G' à n sommets liés aux barres $b(G)$ et $b(G')$ respectivement, nous aurons:

$$b(G) \neq b(G') \iff C(G) \neq C(G'),$$

où $C(G)$ est la représentation canonique de G et $C(G')$ est celle de G'.

Pour calculer la distribution décrite ci-avant pour un générateur de graphes A donné, nous devrons au préalable fixer l, le nombre de graphes à générer, et n, le nombre de sommets de ces graphes (dans la Section 5, nous préciserons comment choisir l). Nous commençons donc par générer l graphes avec le générateur A. Ensuite, nous déterminons la représentation canonique pour chacun de ceux-ci à l'aide du programme **nauty**. Après, nous comptons toutes les occurrences de chaque représentation canonique. Nous nous retrouvons dès lors avec un ensemble D de paires (c, o) dont le premier élément, c, est une représentation canonique et le second, o, est l'occurrence de celle-ci. Pour que cet ensemble constitue la distribution des graphes générés par A, nous devons tenir compte des q graphes qui n'ont pas été générés parmi tous les k graphes d'ordre n différents (pour rappel, deux

graphes G et G' sont différents si et seulement s'ils ne sont pas isomorphes). Pour ce faire, pour chaque graphe G appartenant à ces q graphes, nous ajoutons la paire $(C(G), 0)$ à D, où $C(G)$ est la représentation canonique de G. Nous noterons la distribution des graphes générés par A par $\mathcal{G}(A, n, l)$.

Définition 3.5. Soient A un générateur de graphes aléatoires, l le nombre de graphes à générer et n l'ordre de ceux-ci. Nous disons que A est le *générateur de graphes parfait* si $\mathcal{G}(A, n, l)$ suit une loi uniforme. Nous écrivons celui-ci PGG.

Suite à cette définition, nous avons en fait que les barres du diagramme $\mathcal{G}(\text{PGG}, n, l)$ ont une même hauteur.

Prenons un générateur de graphes A et sa distribution $\mathcal{G}(A, n, l)$. Grâce à celle-ci, nous pouvons comparer ce générateur au générateur de graphes parfait PGG très simplement. Pour ce faire, il suffit d'estimer $\mathcal{G}(\text{PGG}, n, l)$ et de comparer cette distribution à $\mathcal{G}(A, n, l)$. Puisque $\mathcal{G}(\text{PGG}, n, l)$ est uniforme, toutes les barres du diagramme analogue à cette distribution ont une hauteur identique; chaque hauteur peut donc être déterminée comme suit:

$$\frac{1}{k}l,$$

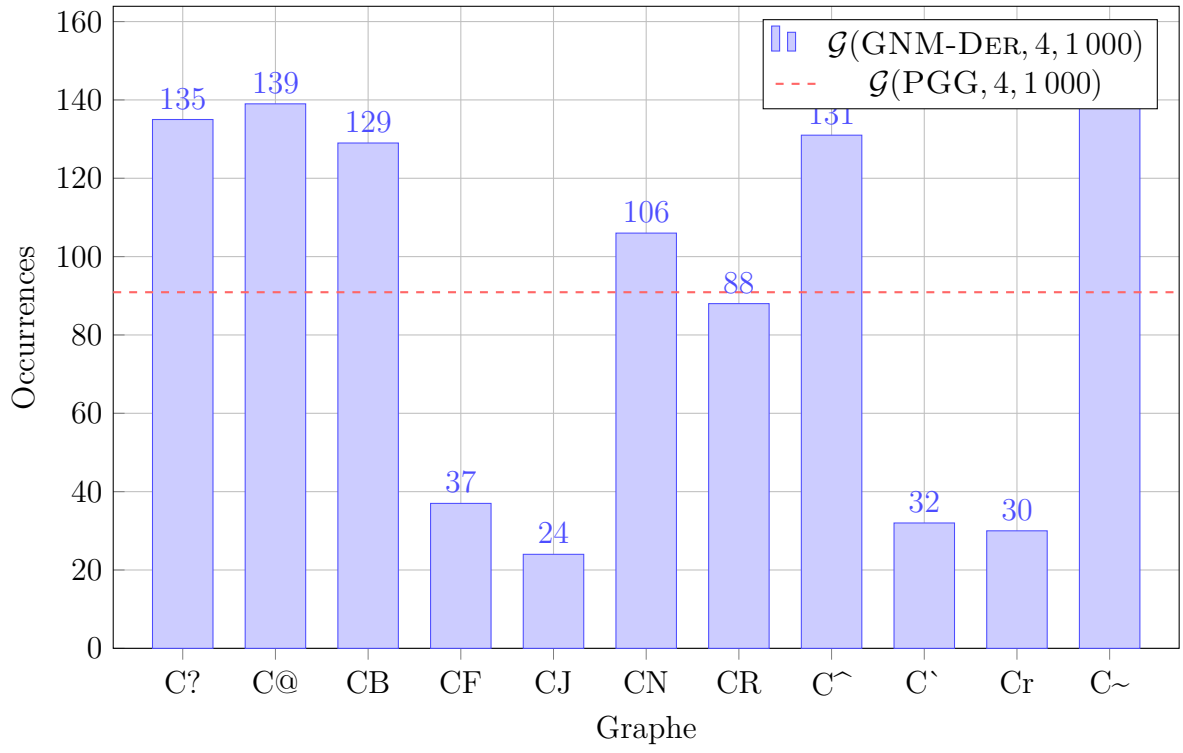
où k est le nombre de graphes différents à n sommets (ou encore, de manière équivalente, le nombre de barres dans le diagramme) et l le nombre de graphes générés lors du calcul de la distribution $\mathcal{G}(A, n, l)$ du générateur A.

Affichons à présent la distribution des graphes générés pour les deux générateurs de graphes GNM-Der et GNP-Der ainsi que celle du générateur de graphes parfait. Fixons le nombre de graphes l à générer à 1 000 et leur ordre n à 4. Les deux distributions obtenues sont montrées dans les Diagrammes 3.1a et 3.1b. Le générateur GNM-Der paraît plus uniforme que le générateur GNP-Der; sa distribution semble être plus proche de celle du générateur de graphes parfait. Fixons désormais le nombre de graphes à générer à 10 000 et le nombre de sommets n de ces graphes à 5. Les deux distributions calculées à partir des deux générateurs sont montrées dans les Diagrammes 3.2a et 3.2b. Le générateur GNP-Der donne l'impression d'être un peu plus efficace que le générateur GNM-Der. En particulier, nous remarquons pour ce dernier que certains graphes sont générés très souvent (900 fois dans notre expérience) et d'autres peu fréquemment (30 fois environ). Toutefois, le générateur GNP-Der génère très rarement (plus ou moins 10 fois) les deux graphes situés aux deux extrémités de la distribution.

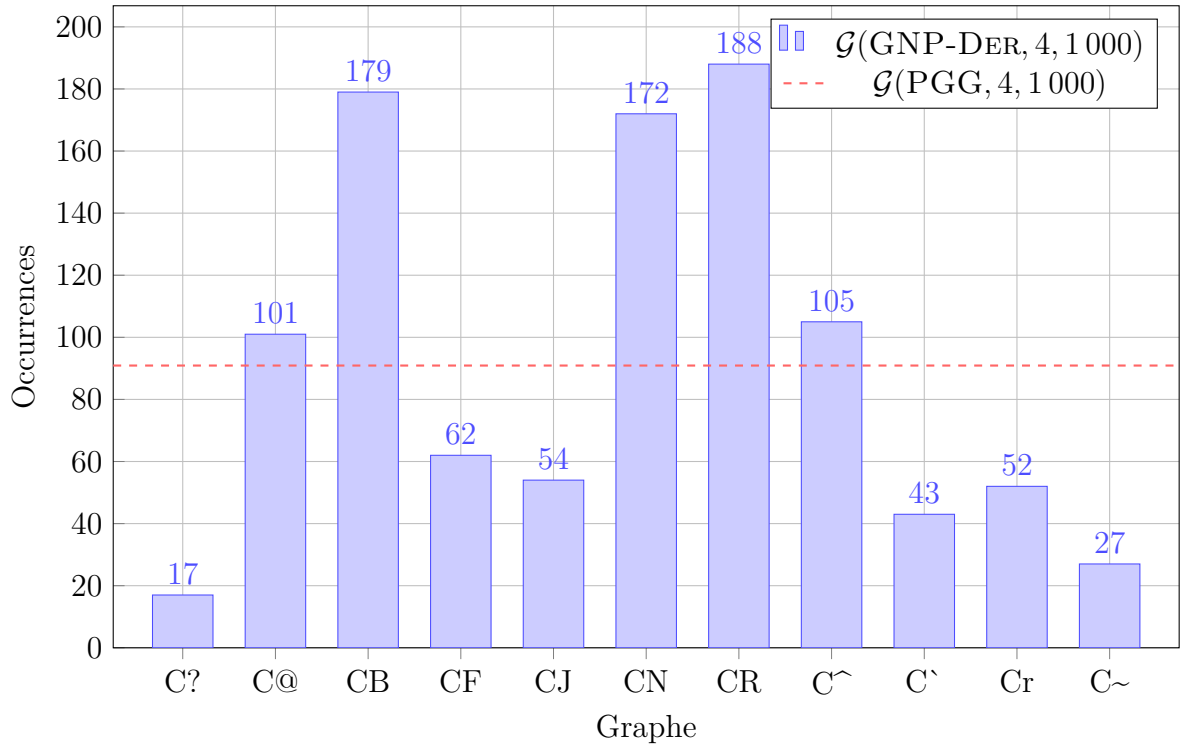
Au vu de ces analyses, nous nous apercevons qu'il est assez difficile et fastidieux de comparer deux générateurs entre eux et de déterminer lequel est meilleur. En conséquence, nous introduirons un évaluateur de générateurs de graphes dans la prochaine section. Celui-ci retournera une valeur numérique représentant la qualité d'un générateur donné. La comparaison entre deux générateurs sera dès lors plus aisée.

3.4 Évaluateur de générateurs de graphes

Précédemment, nous avons vu comment obtenir la distribution $\mathcal{G}(A, n, l)$ des graphes générés par un générateur A. Dans cette section, nous allons nous servir de celle-ci et de la distribution du générateur de graphes parfait afin d'évaluer la qualité du générateur.

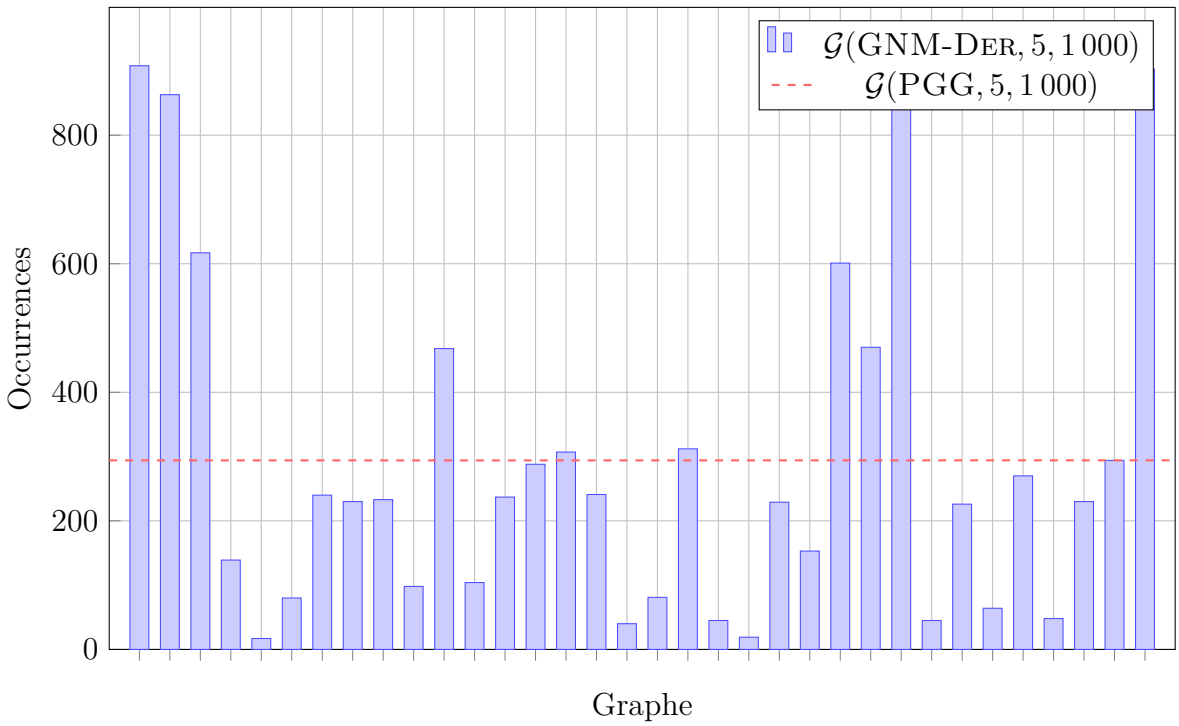


(a) GNM-Der

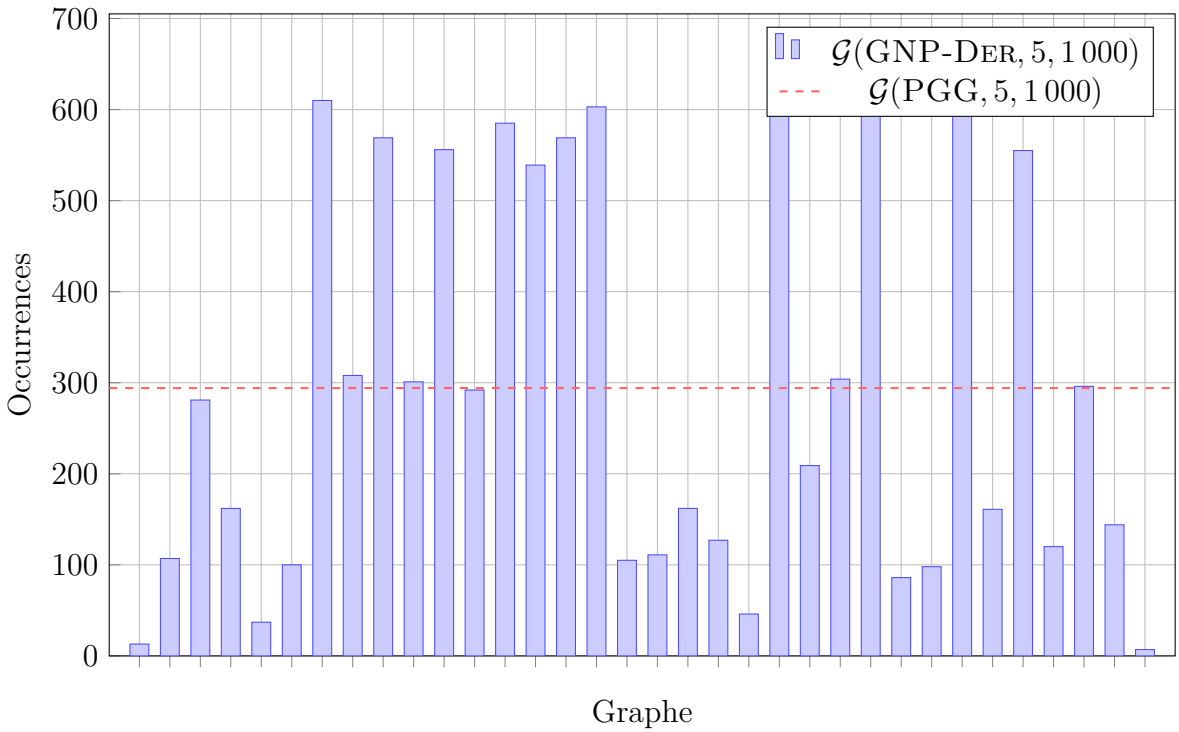


(b) GNP-Der

Diagramme 3.1 – Distribution des graphes générés pour $n = 4$



(a) GNM-Der



(b) GNP-Der

Diagramme 3.2 – Distribution des graphes générés pour $n = 5$

Ordre n des graphes	**Nombre k de graphes différents**
2	2
3	4
4	11
5	34
6	156
7	1 044
8	12 346
9	274 668
10	12 005 168
11	1 018 997 864
12	165 091 172 592
13	50 502 031 367 952
14	29 054 155 657 235 488
15	31 397 381 142 761 241 960
16	63 969 560 113 225 176 176 277

Tableau 3.1 – Nombre de graphes différents en fonction de leur ordre

Pour cela, nous manipulerons ces deux distributions et réaliserons quelques opérations mathématiques dessus. Notons que le nombre k de graphes d'ordre n uniques (d'un point de vue isomorphisme) croît exponentiellement en fonction n. Pour donner une idée de cela, le Tableau 3.1 [3] montre l'évolution de ce nombre pour n allant de 2 à 16. Nous ne pourrons donc malheureusement pas calculer la distribution $\mathcal{G}(A, n, l)$ pour des graphes dont le nombre de sommets n est élevé (supérieur à 10). En effet, cela serait tout bonnement incalculable pour un ordinateur; le temps de calcul serait également exponentiel. Par conséquent, nous évaluerons un générateur de graphes d'ordre faible et extrapolerons l'information pour n'importe quel ordre.

À partir d'un générateur de graphes A et de la distribution $\mathcal{G}(A, n, l)$ de graphes générés par ce générateur, notre évaluateur détermine la qualité de celui-ci en procédant en différentes étapes. Les voici:

1. calculer la fonction de masse f_X du générateur A en normalisant $\mathcal{G}(A, n, l)$;
2. calculer la fonction de masse f_Y du générateur de graphes parfait;
3. pour chaque probabilité de la fonction de masse f_X, calculer la distance entre celle-ci et la probabilité correspondante dans la fonction de masse f_Y;
4. sommer toutes ces distances.

Détaillons cela. Soit la distribution $\mathcal{G}(A, n, l)$ de graphes à n sommets générés par un générateur A. La première étape consiste à calculer la fonction de masse f_X de ce générateur. Il s'agit en réalité d'une normalisation ou plutôt d'une conversation de la

3. Source: http://users.cecs.anu.edu.au/~bdm/data/graphcounts.txt.

hauteur des barres du diagramme $\mathcal{G}(A, n, l)$ en probabilités. Pour réaliser cela, calculons d'abord la somme l des occurrences présentes dans ce diagramme. Cette somme est en fait le nombre de graphes l générés lors du calcul de la distribution $\mathcal{G}(A, n, l)$. Pour chaque barre $b(G)$ du diagramme $\mathcal{G}(A, n, l)$, remplaçons sa hauteur actuelle $h(b(G))$ par:

$$h'(b(G)) = \frac{h(b(G))}{l}.$$

Nous obtenons de ce fait la fonction de masse f_X du générateur A; chaque nouvelle hauteur représente une probabilité. Effectivement, nous pouvons facilement montrer que les trois conditions d'une loi de probabilité sont bien respectées. Plus formellement, pour tout graphe G dans l'ensemble $\{G_1, G_2, \ldots, G_k\}$ qui comprend tous les k graphes à n sommets existants, nous avons:

$$f_X(G) = \mathbb{P}(X = G) = h'(b(G)).$$

Considérons maintenant la fonction de masse f_Y du générateur de graphes parfait. Précisément, il s'agit d'une loi uniforme discrète de paramètre k. Nous avons dès lors que:

$$f_Y(G) = \mathbb{P}(Y = G) = \frac{1}{k},$$

pour tout graphe $G \in \{G_1, G_2, \ldots, G_k\}$. Dans les faits, la fonction f_Y est la normalisation de la distribution $\mathcal{G}(\text{PGG}, n, l)$. Ensuite, pour tout G dans l'ensemble $\{G_1, G_2, \ldots, G_k\}$, nous calculons l'écart (la distance) entre toutes les probabilités de f_X et de f_Y comme ceci:

$$|f_X(G) - f_Y(G)| = \left| h'(b(G)) - \frac{1}{k} \right|.$$

Pour terminer, nous calculons e, la somme de toutes ces distances:

$$e = \sum_G \left| h'(b(G)) - \frac{1}{k} \right| = \sum_G \left| \frac{h(b(G))}{l} - \frac{1}{k} \right|.$$

En conséquence, plus e sera faible et plus la fonction de masse f_X du générateur A sera proche de la fonction de masse f_Y du générateur de graphes parfait. Nous pouvons donc considérer cette valeur e comme une erreur commise par le générateur A. Ainsi, plus e est bas et plus A sera jugé comme meilleur.

Exemple 3.6. Reprenons la distribution $\mathcal{G}(\text{GNP-DER}, 4, 1\,000)$ des graphes générés par GNM-DER présentée dans le Diagramme 3.1a. Dans le cas présent, $n = 4$, $k = 11$ et $l = 1\,000$. Nous pouvons donc calculer la fonction de masse f_X du générateur GNM-DER ainsi que la fonction de masse f_Y du générateur de graphes parfait. Celles-ci sont illustrées dans le Diagramme 3.3. La distance entre chaque probabilité de f_X et de f_Y y est également représentée; chacune est surlignée en vert. Concernant la dernière étape, il suffit de faire la somme de toutes les distances calculées.

Proposition 3.7. *Pour toute valeur e retournée par notre évaluateur de générateurs de graphes, e est toujours comprise dans l'intervalle $[0, 2)$.*

Démonstration. Soient un générateur de graphes A, n l'ordre des graphes générés par celui-ci et k le nombre de graphes différents à n sommets. Vu que e est une somme de

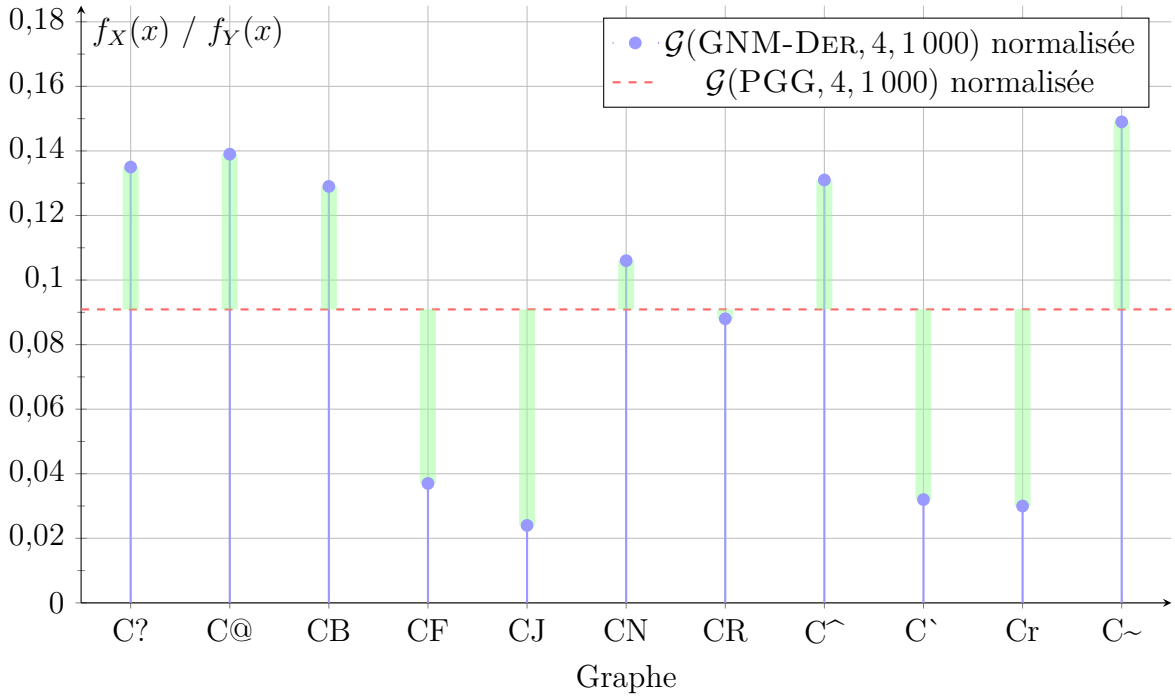


Diagramme 3.3 – Fonction de masse f_X de GNM-Der et fonction de masse f_Y du générateur de graphes parfait

distances, e est trivialement supérieure ou égale à 0. Pour prouver que e est strictement inférieure à 2, considérons le pire générateur de graphes aléatoires possible, c'est-à-dire supposons que A génère toujours le même graphe à n sommets. Cela signifie que la probabilité d'obtenir ce graphe est de 1 et la probabilité d'obtenir un des $k-1$ autres graphes est de 0. Ceci définit la fonction de masse f_X du générateur A. Calculons maintenant e, la somme des distances entre chaque probabilité de f_X et de la fonction de masse f_Y du générateur de graphes parfait:

$$\begin{aligned} e &= 1\left|1-\frac{1}{k}\right| + (k-1)\left|0-\frac{1}{k}\right| \\ &= \left|1-\frac{1}{k}\right| + (k-1)\left|-\frac{1}{k}\right| \\ &= 1-\frac{1}{k}+(k-1)\frac{1}{k} && (k \geqslant 1) \\ &= 1-\frac{1}{k}+1-\frac{1}{k} \\ &= 2-\frac{2}{k} \\ &< 2. && (k > 0) \end{aligned}$$

Pour tout générateur de graphes aléatoires, nous avons dès lors que la valeur e retournée par notre évaluateur se situe dans l'intervalle $[0, 2)$. □

Exemple 3.8. Le Diagramme 3.4 montre la fonction de masse f_X d'un des pires générateurs de graphes à 4 sommets, noté A.

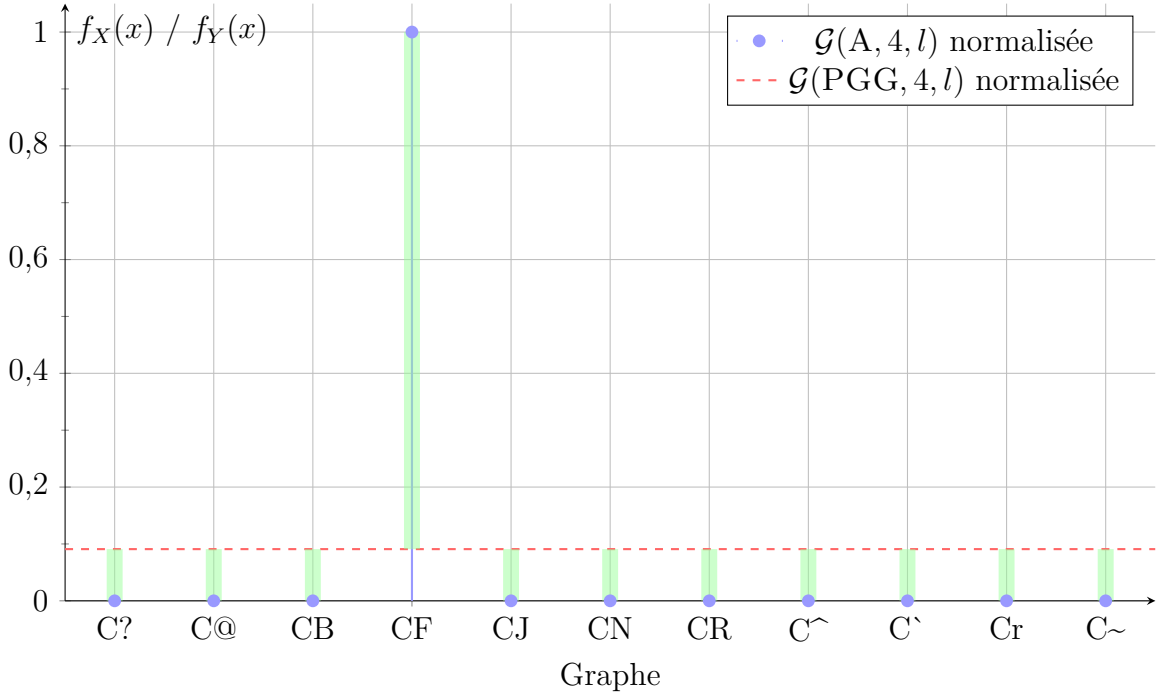


Diagramme 3.4 – Fonction de masse f_X d'un des pires générateurs A et fonction de masse f_Y du générateur de graphes parfait

Dans la section suivante, nous concevrons notre propre générateur de graphes aléatoires à partir de la distribution de la taille des graphes d'ordre n.

4 Création d'un générateur de graphes aléatoires

Dans la section précédente, nous avons présenté quelques générateurs de graphes aléatoires ainsi qu'une manière d'évaluer la qualité de ceux-ci. Nous allons présentement concevoir notre propre générateur de graphes.

Nous commencerons par analyser la distribution de la taille de tous les graphes distincts à n sommets. Plus précisément, nous générerons tous les graphes uniques d'ordre n et calculerons la distribution du nombre d'arêtes de ces graphes. Nous obtiendrons par conséquent un diagramme à barres où chacune correspondra à un nombre d'arêtes m et dont sa hauteur représentera la quantité de graphes différents à m arêtes. Par exemple, pour $n = 6$, nous pouvons d'avance affirmer qu'il existe davantage de graphes à quatre arêtes que de graphes à zéro arête (il existe d'ailleurs qu'un seul graphe à zéro arête). Un bon générateur devrait donc générer en plus grande quantité des graphes à quatre arêtes que des graphes à zéro arête. Faisons remarquer que le générateur GNM-Der ne procède pas comme cela; la probabilité que celui-ci génère un graphe contenant quatre arêtes est égale à la probabilité qu'il génère un graphe avec zéro arête. Il serait dès lors possible d'améliorer ce générateur juste en modifiant le calcul du nombre d'arêtes; il suffirait de tenir compte de la distribution de la taille des graphes à n sommets lors de ce calcul. C'est ce que nous ferons dans cette section.

Par conséquent, nous analyserons dans une première étape la distribution de la taille de tous les graphes différents d'ordre n. Dans une seconde étape, étant donné un nombre de sommets n quelconque, nous estimerons avec le plus de précision possible cette distribution. Nous réaliserons dans une dernière étape notre générateur de graphes aléatoires basé sur le modèle $G(n, m)$.

4.1 Analyse de la distribution de la taille des graphes

Nous allons à présent calculer la distribution de la taille des graphes à n sommets avec n fixé. Nous allons l'examiner attentivement et tenterons de découvrir quelle loi de probabilité s'en rapproche le plus. Pour réaliser cela, nous emploierons le test d'ajustement du khi-deux décrit dans la Section 2.2.

Posons $\mathcal{S}(n)$, la distribution de la taille de tous les graphes différents à n sommets. La procédure pour déterminer celle-ci est assez simple. Soit n l'ordre des graphes. Nous générons tout d'abord tous les graphes différents à n sommets à l'aide du programme nauty [1]; supposons qu'il y en existe k. Ensuite, nous déterminons la taille de chaque graphe généré. Enfin, nous comptons l'occurrence de chaque taille et construisons ainsi $\mathcal{S}(n)$. De la même manière que dans la Section 3.3, nous pouvons représenter cette distribution comme un diagramme à barres dans lequel chacune correspond à une taille de graphes et sa hauteur désigne la quantité de graphes ayant cette taille. Rappelons que la taille maximale $\gamma(n)$ d'un graphe à n sommets est donnée par l'équation suivante:

$$\gamma(n) = \frac{n(n-1)}{2}.$$

1. La documentation de ce programme peut être trouvée à l'adresse suivante: http://pallini.di.uniroma1.it/Guide.html.

De ce fait, le diagramme résultant de $\mathcal{S}(n)$ possédera exactement $\gamma(n)+1$ barres. Effectivement, la taille d'un graphe varie entre 0 et $\gamma(n)$ compris. Soulignons le fait que la distribution $\mathcal{S}(n)$ ne peut pas être calculée pour n'importe quel ordre n. La raison à cela est simple: dans la méthode décrite ci-avant, nous générons tous les k graphes différents à n sommets; comme k est exponentiel en fonction de n, $\mathcal{S}(n)$ ne peut être obtenue en un temps raisonnable quand n est grand. Essayons donc de trouver une loi de probabilité permettant d'estimer celle-ci précisément.

Pour cela, calculons et observons la distribution $\mathcal{S}(n)$ pour différents n. Cela est montré dans les Diagrammes 4.1a à 4.1d pour $n \in \{5,6,7,8\}$. Nous constatons que chaque distribution ressemble à une loi normale et à une loi binomiale. Vérifions cela graphiquement.

Nous devons pour ce faire estimer les paramètres de ces deux lois.

Commençons par la loi normale $\mathcal{N}(\mu,\sigma)$. Celle-ci possède deux paramètres μ et σ et peuvent être estimés facilement en calculant la moyenne empirique et l'écart-type de nos données, à savoir l'ensemble des k tailles de graphes déterminées lors du calcul de $\mathcal{S}(n)$. Nous pouvons calculer $\hat{\mu}$ et $\hat{\sigma}$, l'estimation de μ et de σ respectivement, comme suit:

$$\hat{\mu} = \frac{1}{k}\sum_{i=1}^{k} m_i, \tag{4.1}$$

$$\hat{\sigma} = \sqrt{\frac{1}{k}\sum_{i=1}^{k}(m_i-\hat{\mu})^2}, \tag{4.2}$$

où k est le nombre de données et m_i la i^{e} donnée.

Proposition 4.1. *Soient $\hat{\mu}$ et $\hat{\sigma}$ l'estimation de la moyenne et de l'écart-type des données respectivement. Alors les deux paramètres r et p d'un loi binomiale $\mathcal{B}(r,p)$ peuvent être estimés par:*

$$\hat{r} = \left\lfloor \frac{\hat{\mu}^2}{\hat{\mu}-\hat{\sigma}^2} \right\rceil, \tag{4.3}$$

$$\hat{p} = 1 - \frac{\hat{\sigma}^2}{\hat{\mu}}. \tag{4.4}$$

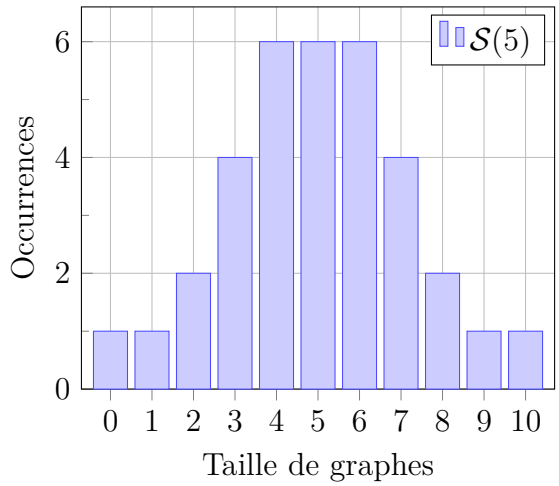
Passons à l'estimation des paramètres de la loi binomiale $\mathcal{B}(r,p)$.

Démonstration. Nous avons vu dans la Section 2.2 que la moyenne μ et l'écart-type σ d'une loi binomiale $\mathcal{B}(r,p)$ sont:

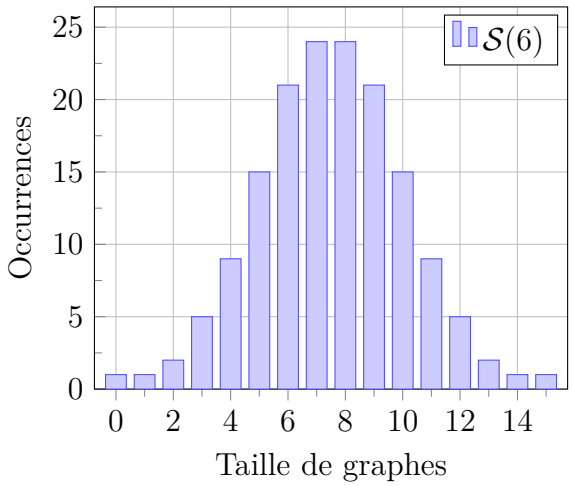
$$\mu = rp \quad \text{et} \quad \sigma = \sqrt{rp(1-p)}.$$

Soient $\hat{\mu}$, l'estimation de la moyenne de nos données, et $\hat{\sigma}$, l'estimation de l'écart-type de celles-ci. Ces estimations sont effectuées avec les Équations (4.1) et (4.2). Nous nous retrouvons donc avec un système de deux équations à deux inconnues. Résolvons-le.

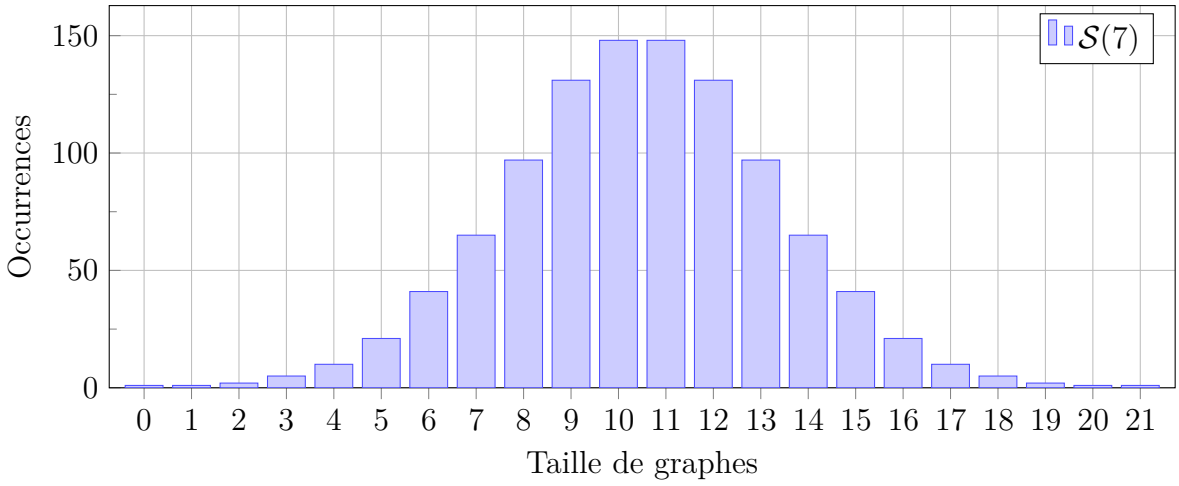
$$\begin{cases} \hat{\mu} = \hat{r}\hat{p} \\ \hat{\sigma} = \sqrt{\hat{r}\hat{p}(1-\hat{p})} \end{cases} \iff \begin{cases} \hat{\mu} = \hat{r}\hat{p} \\ \hat{\sigma} = \sqrt{\hat{\mu}(1-\hat{p})} \end{cases}$$



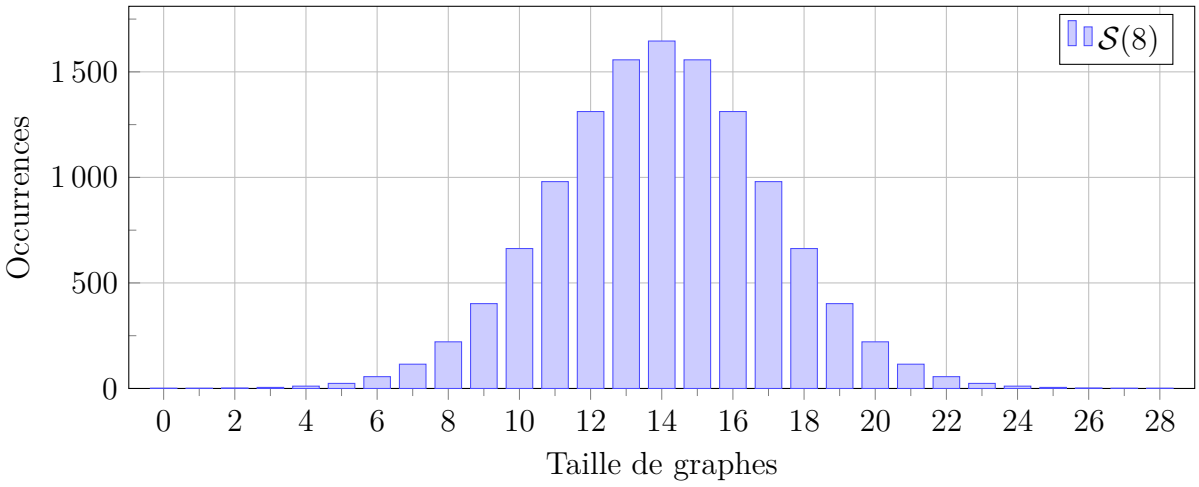
(a) $n = 5$



(b) $n = 6$



(c) $n = 7$



(d) $n = 8$

Diagramme 4.1 – Distributions $\mathcal{S}(n)$

$$
\begin{aligned}
&\iff \begin{cases} \hat{\mu} = \hat{r}\hat{p} \\ \hat{p} = 1 - \dfrac{\hat{\sigma}^2}{\hat{\mu}} \end{cases} \\
&\iff \begin{cases} \hat{\mu} = \hat{r}\left(1 - \dfrac{\hat{\sigma}^2}{\hat{\mu}}\right) \\ \hat{p} = 1 - \dfrac{\hat{\sigma}^2}{\hat{\mu}} \end{cases} \\
&\iff \begin{cases} \hat{r} = \dfrac{\hat{\mu}}{1 - \hat{\sigma}^2/\hat{\mu}} \\ \hat{p} = 1 - \dfrac{\hat{\sigma}^2}{\hat{\mu}} \end{cases} \\
&\iff \begin{cases} \hat{r} = \dfrac{\hat{\mu}^2}{\hat{\mu} - \hat{\sigma}^2} \\ \hat{p} = 1 - \dfrac{\hat{\sigma}^2}{\hat{\mu}} \end{cases}
\end{aligned}
$$

Puisque le paramètre r d'une loi binomiale $\mathcal{B}(r, p)$ est un entier positif, nous devons arrondir $\hat{r}$ à l'entier le plus proche. Donc,

$$\hat{r} = \left\lfloor \frac{\hat{\mu}^2}{\hat{\mu} - \hat{\sigma}^2} \right\rceil \quad \text{et} \quad \hat{p} = 1 - \frac{\hat{\sigma}^2}{\hat{\mu}}.$$

□

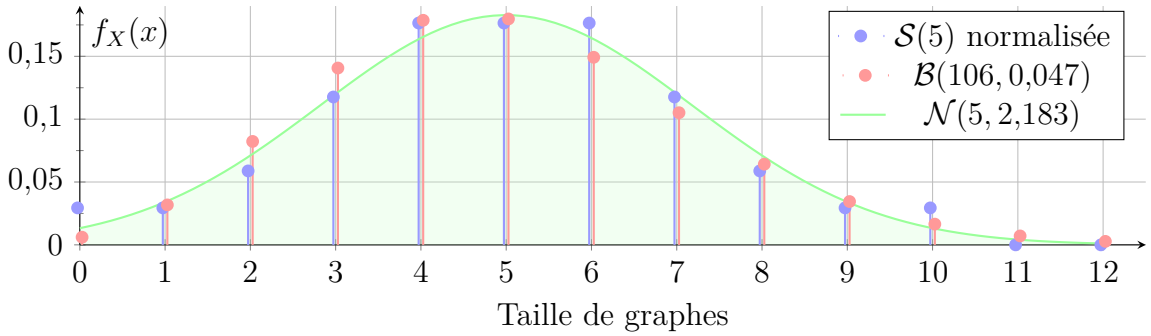
Nous pouvons maintenant comparer chaque distribution $\mathcal{S}(n)$ du Diagramme 4.1 avec la loi normale et la loi binomiale correspondantes. Pour faire cela, normalisons d'abord cette distribution en calculant sa fonction de masse de la manière suivante:

1. calculer la somme k des occurrences de $\mathcal{S}(n)$;
2. diviser chaque occurrence par k.

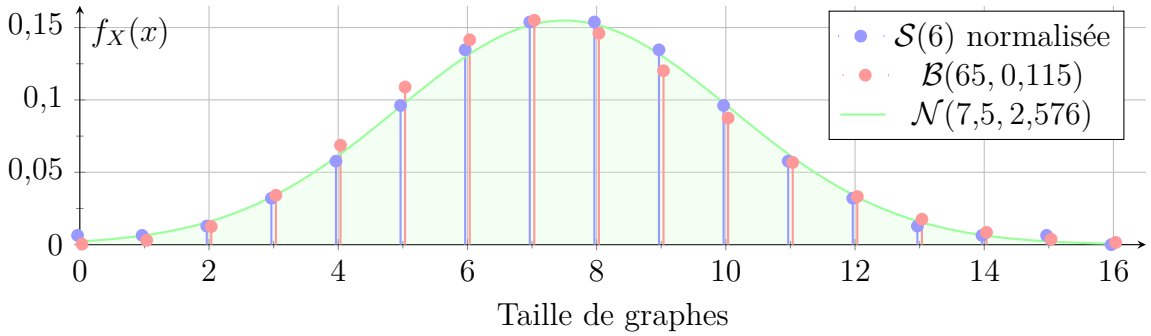
Ensuite, déterminons les paramètres de la loi normale et de la loi binomiale correspondantes à l'aide des Équations (4.1) à (4.4). Grâce à ces paramètres, nous pouvons calculer la densité de probabilité de la loi normale et la fonction de masse de la loi binomiale. Affichons cette densité de probabilité, cette fonction de masse ainsi que la fonction de masse de $\mathcal{S}(n)$; cela est montré dans les Diagrammes 4.2a à 4.2d. Nous remarquons que ces trois fonctions sont proches pour chaque ordre n. Par conséquent, la distribution $\mathcal{S}(n)$ donne l'impression de suivre une loi normale et une loi binomiale.

Confirmons cela statistiquement au moyen du test d'ajustement du khi-deux. Pour chaque distribution $\mathcal{S}(n)$, pour n allant de 5 à 8, nous allons effectuer deux tests de ce type: un pour vérifier que nos données suivent une loi normale et un autre pour tester si celles-ci suivent une loi binomiale.

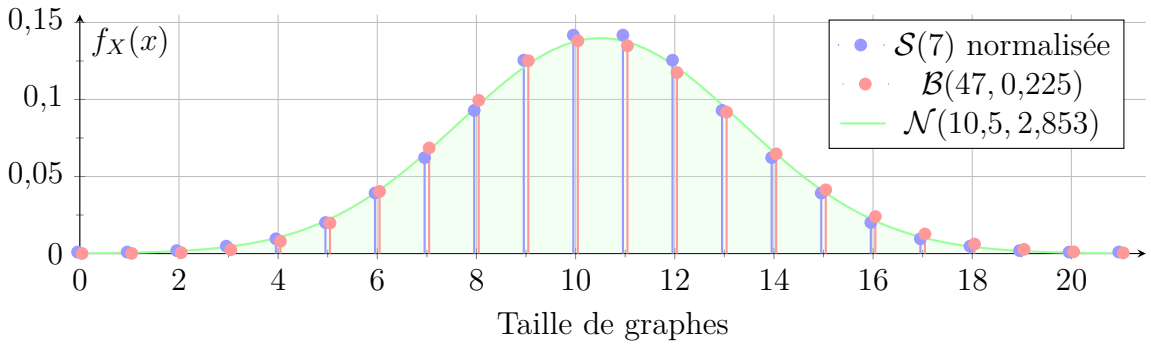
Pour ce premier test, calculons la densité de probabilité f_X de la loi normale analogue à $\mathcal{S}(n)$. Notons X sa variable aléatoire. Pour le second test, déterminons la fonction de masse f_Y de la loi binomiale correspondante à $\mathcal{S}(n)$, avec Y la variable aléatoire de cette loi.



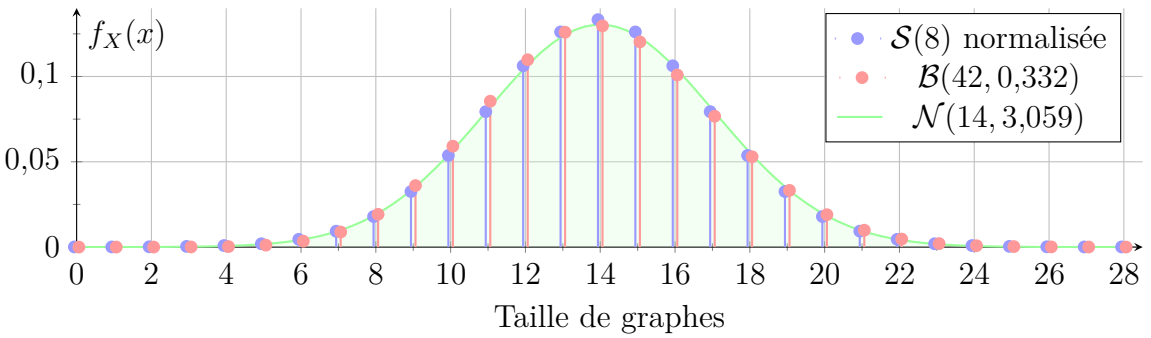
(a) $n = 5$



(b) $n = 6$



(c) $n = 7$



(d) $n = 8$

Diagramme 4.2 – Fonction de masse de la taille des graphes d'ordre n

Ordre des graphes	***p*-valeur obtenue**	
	Loi normale	**Loi binomiale**
5	0,96322	0,92122
6	1	0,99961
7	0,99983	0,99009
8	0,84236	0,00766

Tableau 4.1 – Résultats des tests d'ajustement du khi-deux

Les classes sont naturellement créées: chacune représente une taille de graphes précise. Nous pouvons donc raisonnablement supposer que la c^{e} classe coïncide avec une taille de graphes égale à c. L'effectif théorique $T(A_c)$ de chaque classe A_c peut être calculé comme suit:

$$T(A_c) = k\mathbb{P}(Z \in A_c),$$

où k est la somme de toutes les occurrences de $\mathcal{S}(n)$, $Z \in \{X, Y\}$ et

$$\mathbb{P}(Z \in A_c) = \begin{cases} F_X(c+0{,}5) - F_X(c-0{,}5) & \text{si } Z = X, \\ f_Y(c) = \mathbb{P}(Y = c) & \text{si } Z = Y, \end{cases}$$

où F_X est la fonction de répartition calculée à partir de f_X (voir l'Équation (2.1)). En ce qui concerne l'effectif observé $O(A_c)$ de chaque classe A_c, il est égal à l'occurrence de la taille de graphes c; cela peut être trouvé dans la distribution $\mathcal{S}(n)$. Dans le mesure où le test d'ajustement du khi-deux peut se montrer invalide en présence d'effectifs théoriques inférieurs à 5, nous supprimons les deux effectifs $T(A_c)$ et $O(A_c)$ pour toute classe A_c telle que $T(A_c) < 5$.

Au moyen des effectifs théoriques T et des effectifs observés O, nous pouvons mesurer la statistique de test et par la suite la p-valeur. Pour $n \in \{5, 6, 7, 8\}$, les résultats des deux tests d'ajustement du khi-deux sont présentés dans le Tableau 4.1. Nous y trouvons la p-valeur pour la loi normale tout comme la p-valeur pour la loi binomiale. Nous nous apercevons que cette première est plus élevée que cette seconde. De ce fait, nos données semblent suivre davantage une loi normale qu'une loi binomiale. Il est donc plus judicieux d'utiliser celle-ci pour estimer $\mathcal{S}(n)$.

Dans la prochaine section, nous tenterons dès lors d'estimer les paramètres de la loi normale analogue à $\mathcal{S}(n)$ pour n'importe quel ordre de graphes n.

4.2 Approximation de la distribution de la taille des graphes

Dans cette section, nous allons tenter d'estimer $\mathcal{S}(n)$, la distribution de la taille des graphes d'ordre n, pour un n quelconque. Plus précisément, nous estimerons les deux paramètres de la loi normale correspondante à $\mathcal{S}(n)$. Notons celle-ci $\mathcal{N}(\mu_n, \sigma_n)$.

Pour cela, déterminons d'abord $\hat{\mu}_n$ et $\hat{\sigma}_n$, l'estimation des deux paramètres μ_n et σ_n de $\mathcal{N}(\mu_n, \sigma_n)$ respectivement, pour différents n. Ce calcul est réalisé en utilisant les

Équations (4.1) et (4.2) vues dans la section précédente. Ensuite, affichons l'évolution de $\hat{\mu}_n$ et de $\hat{\sigma}_n$ en fonction de n. Enfin, tentons de découvrir une tendance se dégageant des diagrammes résultants.

Dans la Section 4.1, nous avons fait remarquer qu'il était difficile de calculer $\mathcal{S}(n)$, et par conséquent $\mathcal{N}(\mu_n, \sigma_n)$, $\hat{\mu}_n$ et $\hat{\sigma}_n$, lorsque n est élevé ($n > 10$) avec notre méthode. Néanmoins, nous avons trouvé un site web[2] répertoriant les distributions $\mathcal{S}(n)$ pour n allant de 1 à 16. À partir de celles-ci, $\hat{\mu}_n$ et $\hat{\sigma}_n$ peuvent être aisément calculés. Grâce à cela, notre approximation de $\mathcal{S}(n)$ pour tout n devrait être plus précise vu que nous avons davantage de données (16 au lieu de 10).

Les Diagrammes 4.3 et 4.4 montrent respectivement l'évolution de $\hat{\mu}_n$ et de $\hat{\sigma}_n$ en fonction du nombre de sommets n. En ce qui concerne $\hat{\mu}_n$, nous constatons que cette valeur est donnée par la moitié de la taille maximale $\gamma(n)$ d'un graphe, c'est-à-dire:

$$\hat{\mu}_n = \frac{\gamma(n)}{2} = \frac{n(n-1)}{4}, \forall n \in \mathbb{N}_{\geqslant 1}. \tag{4.5}$$

Effectivement, comme la distribution $\mathcal{S}(n)$ s'étend de 0 à $\gamma(n)$ et que celle-ci est une fonction symétrique (voir les Diagrammes 4.1a à 4.1d présents à la Page 36), nous avons que $\hat{\mu}_n = \gamma(n)/2$. Quant à $\hat{\sigma}_n$, nous nous apercevons que les valeurs semblent linéaires. En conséquence, nous pouvons raisonnablement effectuer une régression linéaire simple dont le but est de trouver une droite $\tilde{\sigma}(n) = \beta_0 + \beta_1 n$ prédisant *le mieux possible* $\hat{\sigma}_n$. Cette droite $\tilde{\sigma}(n)$ est en fait une approximation de $\hat{\sigma}_n$. En appliquant les Équations (2.2) et (2.3) de la Section 2.2, nous trouvons que:

$$\beta_0 = 0{,}1575828904704868 \quad \text{et} \quad \beta_1 = 0{,}3375899521271764.$$

Nous avons en résumé ceci pour $\hat{\sigma}_n$ et $\tilde{\sigma}(n)$:

$$\hat{\sigma}_n \approx \tilde{\sigma}(n) = 0{,}3375899521271764n + 0{,}1575828904704868, \forall n \in \mathbb{N}_{\geqslant 1}. \tag{4.6}$$

La droite $\tilde{\sigma}(n)$ est affichée dans le Diagramme 4.4.

Nous venons donc de trouver une approximation de $\mathcal{S}(n)$ étant donné que nous pouvons calculer avec exactitude $\hat{\mu}_n$ et mesurer $\hat{\sigma}_n$ précisément pour tout n. Dans la section suivante, nous utiliserons cette approximation dans l'objectif de créer notre propre générateur de graphes aléatoires.

4.3 Conception du générateur de graphes

Dans cette section, nous allons construire notre propre générateur de graphes aléatoires à partir des éléments découverts dans les sections d'avant. Ce générateur sera basé sur l'approximation de la distribution $\mathcal{S}(n)$ de la taille des graphes. Comme celle-ci est en réalité l'approximation de la loi normale correspondante à $\mathcal{S}(n)$, nous appelons ce nouveau générateur GNM-Norm.

Au même titre que le générateur GNM-Der, GNM-Norm est un dérivé du modèle $G(n, m)$ décrit dans la Section 3.1. De ce fait, il reçoit également en entrée une

2. http://users.cecs.anu.edu.au/~bdm/data/graphcounts.txt.

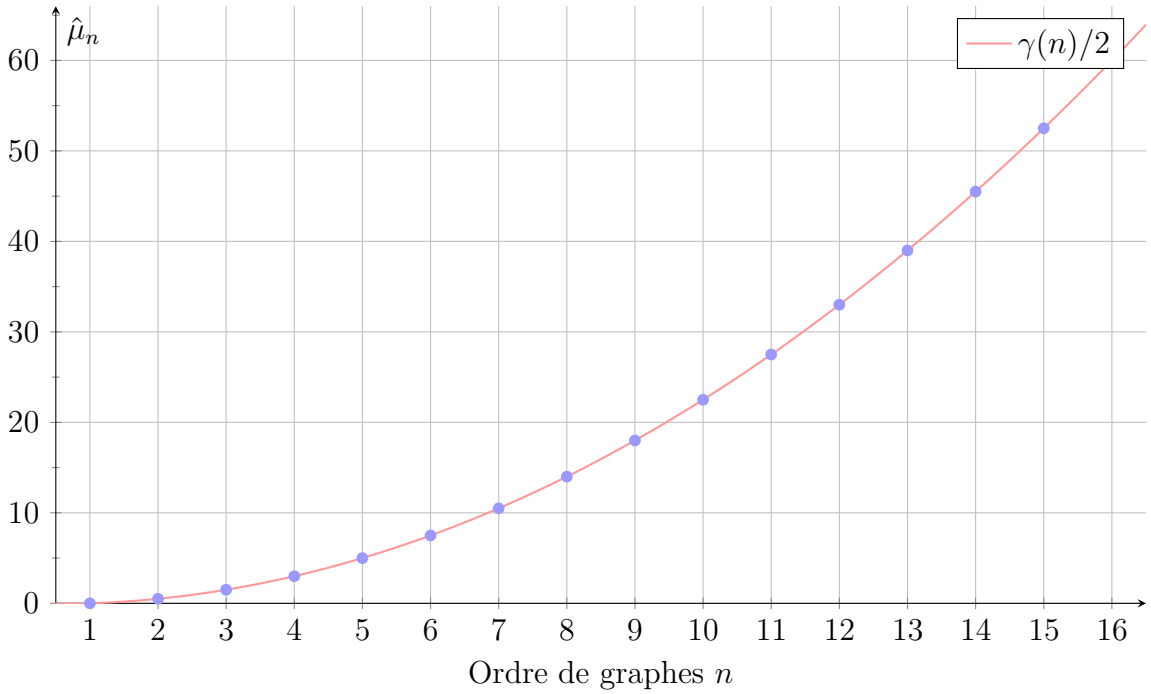


Diagramme 4.3 – Évolution de $\hat{\mu}_n$ en fonction de l'ordre de graphes n

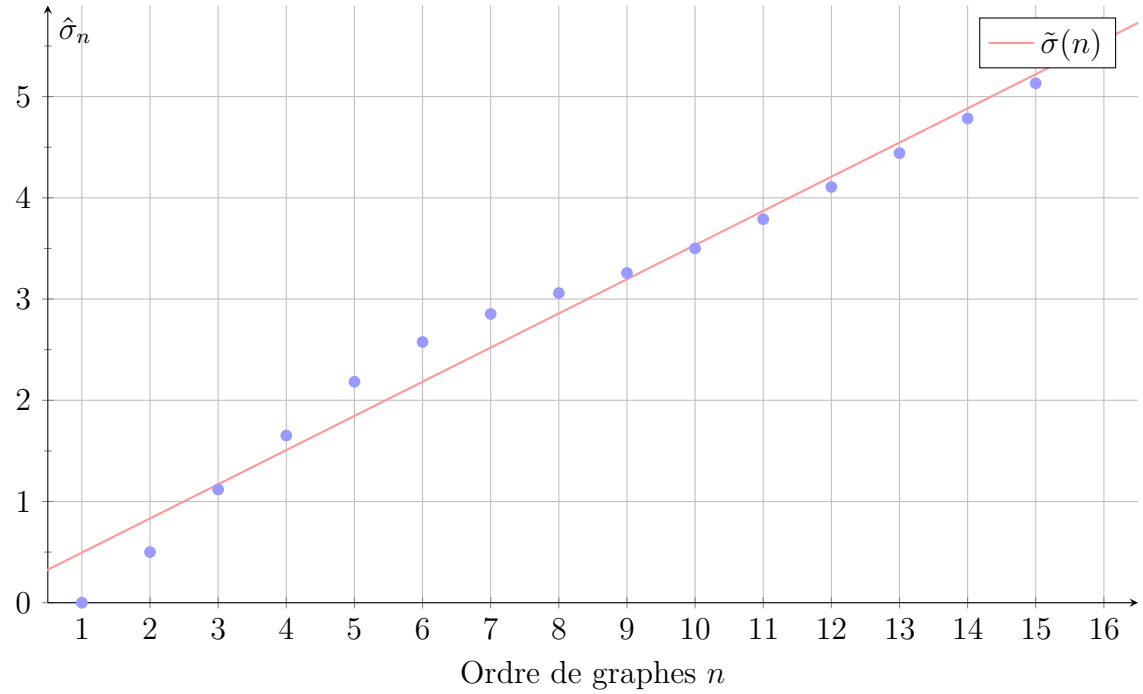


Diagramme 4.4 – Évolution de $\hat{\sigma}_n$ en fonction de l'ordre de graphes n

Algorithme 4.1 GNM-Norm(n)

Entrée:
L'ordre n du graphe à générer.
Sortie:
Un graphe aléatoire à n sommets.

1: $M \leftarrow n(n-1)/2$
2: $\mu \leftarrow M/2$
3: $\sigma \leftarrow 0{,}3375899521271764n + 0{,}1575828904704868$
4: $m \leftarrow$ Générer un nombre aléatoire suivant la loi normale $\mathcal{N}(\mu, \sigma)$
5: **si** $m < 0$ **alors**
6: $\quad m \leftarrow 0$
7: **sinon si** $m > M$ **alors**
8: $\quad m \leftarrow M$
9: **sinon**
10: $\quad m \leftarrow \lfloor m \rceil$
11: **retourner** GNM(n, m)

valeur n représentant l'ordre du graphe à générer. En premier lieu, nous calculons la taille maximale $\gamma(n)$ d'un graphe possédant n sommets ainsi que les estimations $\hat{\mu}_n$ et $\hat{\sigma}_n$ des deux paramètres de la loi normale analogue à $\mathcal{S}(n)$. Les deux paramètres $\hat{\mu}_n$ et $\hat{\sigma}_n$ sont déterminés *via* les Équations (4.5) et (4.6). Nous générons en second lieu un nombre aléatoire m suivant la loi normale $\mathcal{N}(\hat{\mu}_n, \hat{\sigma}_n)$. En troisième lieu, nous calculons m' comme suit:

$$m' = \begin{cases} 0 & \text{si } m < 0, \\ \gamma(n) & \text{si } m > \gamma(n), \\ \lfloor m \rceil & \text{sinon.} \end{cases}$$

Cette étape est nécessaire puisque la taille d'un graphe est un entier compris entre 0 et $\gamma(n)$. En dernier lieu, nous générons un graphe en appelant l'Algorithme 3.1 avec n et m' comme paramètres. L'Algorithme 4.1 est une implémentation de l'algorithme décrit dans ce paragraphe.

Nous venons donc de concevoir un générateur de graphes aléatoires à n sommets dont le nombre d'arêtes suit une approximation de la distribution $\mathcal{S}(n)$. Dans la prochaine section, nous parlerons de quelques détails d'implémentation et réaliserons une comparaison entre les trois générateurs de graphes GNM-Der, GNP-Der et GNM-Norm.

5 Mise en œuvre

Nous allons désormais mettre en œuvre tout ce que nous avons vu au long de ce travail. Pour cela, nous commencerons par discuter de notre implémentation et des divers choix effectués. Ensuite, nous comparerons tous les générateurs de graphes aléatoires que nous avons étudiés, à savoir GNM-Der, GNP-Der et GNM-Norm. Nous réaliserons cette comparaison dans le but de trouver quel générateur est le plus uniforme et donc de découvrir celui qui se rapproche le plus du générateur de graphes parfait. Pour cela, nous emploierons notre évaluateur de générateurs de graphes présenté dans la Section 3.4.

5.1 Implémentation

Dans cette section, nous parlerons de notre implémentation, en particulier des outils utilisés, des problèmes rencontrés ainsi que des optimisations pratiquées. Celle-ci peut être trouvée sur le site GitHub [1].

Commençons par les outils que nous avons eu recours durant ce mémoire. Nous avons décidé de réaliser toute notre implémentation dans le langage de programmation Python [2]. Celui-ci dispose d'une syntaxe légère et permet de faire une grande quantité de choses en quelques lignes de code seulement. Python a également l'avantage de posséder un immense catalogue de bibliothèques logicielles (*librairies* en anglais) scientifiques. En contrepartie, ce langage est plus lent à l'exécution que beaucoup d'autres. Concernant les bibliothèques logicielles dont nous avons fait usage, elles sont citées et décrites ci-dessous.

— NetworkX [3]: outils permettant de créer, de générer et de manipuler des graphes.

— NumPy [4]: offre la possibilité de manipuler très facilement énormément de données. Étant écrites en C en interne, les routines proposées par NumPy sont d'une rapidité exemplaire.

— SciPy [5]: contient une quantité généreuse d'utilitaires probabilistes et statistiques.

— Matplotlib [6]: permet de produire plusieurs types de diagrammes.

Comme précisé dans la Section 3.2, nous avons tiré profit du programme nauty pour résoudre le problème d'isomorphisme de graphes. Celui-ci est un des plus rapides qu'il soit, si ce n'est le plus rapide, pour ce problème. La communication entre nauty et notre code Python est assurée par des fichiers temporaires. Concernant la rédaction de ce document, nous avons utilisé LaTeX [7]. En particulier, les figures et les diagrammes ont été générés par TikZ et PGFPlots, deux packages compris dans LaTeX.

Les deux principaux problèmes que nous avons dû faire face portent sur le temps de calcul et sur l'espace mémoire.

Premièrement, le calcul de la distribution des graphes générés par un générateur peut très rapidement devenir long. Effectivement, lors de ce calcul, nous générerons l graphes

1. Lien URL de notre implémentation: https://github.com/Nico-Salamone/graphgen.
2. https://www.python.org.
3. https://networkx.github.io.
4. https://www.numpy.org.
5. https://www.scipy.org.
6. https://matplotlib.org.
7. https://www.latex-project.org.

à n sommets, où l doit être élevé. Idéalement, l devrait être au moins aussi grand que k, le nombre de graphes différents d'ordre n, afin de laisser une chance au générateur de générer ceux-ci au moins une fois. Puisque k est exponentiel en fonction de n, l devrait l'être également. Malheureusement, cela est inhérent au problème; il est donc impossible à résoudre. Pareillement, notre évaluateur de générateurs de graphes souffre de ce problème vu qu'il est basé sur la distribution des graphes générés. Néanmoins, nous pouvons réduire le temps de ce calcul d'un certain facteur h. Pour cela, nous générons les l graphes en parallèle sur le nombre de processeurs h disponibles sur la machine exécutant le programme (chacun génère environ l/h graphes).

Deuxièmement, une autre difficulté s'est posée; cette fois-ci au niveau de la mémoire. En effet, il est impossible de stocker en mémoire les l graphes générés dans la mesure où l est exponentiel. Pour pallier ce problème, nous employons des générateurs Python qui génèrent des données à la demande. Cela signifie que dès qu'un graphe est généré, celui-ci est traité et aussitôt jeté.

Troisièmement, l'espace disque pris par nos fichiers temporaires, servant à la communication entre nauty et notre programme, pouvait atteindre quelques gigaoctets en peu de temps. Nous avons remédié cela en générant les graphes morceau par morceau (*chunk*). De cette façon, au lieu de générer l/h graphes d'un coup par processeur, nous ne générons qu'au maximum l' graphes à la fois. Dans notre implémentation, nous avons fixé l' à 1 000 000. Grâce à cela, la taille de l'ensemble des fichiers temporaires ne dépasse pas les $h \times 100$ mégaoctets.

Dernièrement, le temps de calcul de l'estimation des paramètres de la loi normale et de la loi binomiale approchant $\mathcal{S}(n)$ pouvait être important (rappelons que $\mathcal{S}(n)$ est la distribution de la taille des k graphes existants d'ordre n). Effectivement, nous estimions la moyenne μ et l'écart-type σ par les Équations (4.1) et (4.2) dans lesquelles nous sommons k éléments avec k exponentiel en n. À la place d'utiliser celles-ci, nous pouvons calculer la moyenne pondérée et l'écart-type pondéré. Pour cela, nous calculons d'abord la distribution $\mathcal{S}(n)$. Cette dernière contient l'occurrence w_i de chaque taille de graphes m_i. Cette occurrence w_i peut en fait être vue comme un poids. Remarquons que:

$$k = \sum_{i=1}^{\gamma(n)+1} w_i,$$

où $\gamma(n)$ est le nombre maximal d'arêtes qu'un graphe d'ordre n peut posséder. La moyenne pondérée et l'écart-type pondéré [7] peuvent dès lors être calculés comme suit:

$$\hat{\mu} = \frac{1}{k} \sum_{i=1}^{\gamma(n)+1} w_i m_i,$$

$$\hat{\sigma} = \sqrt{\frac{1}{k} \sum_{i=1}^{\gamma(n)+1} w_i (m_i - \hat{\mu})^2},$$

où $\hat{\mu}$ est l'estimation de μ et $\hat{\sigma}$ est l'estimation de σ. Avec ceci, nous ne sommons plus que $\gamma(n) + 1$ éléments au lieu de k.

5.2 Comparaison des générateurs de graphes

Nous allons dans cette section comparer les trois générateurs de graphes GNM-Der, GNP-Der et GNM-Norm à l'aide de notre évaluateur de générateurs. Pour rappel, celui-ci calcule l'erreur e commise par un générateur A en comparant la fonction de masse f_X de celui-ci et la fonction de masse f_Y du générateur de graphes parfait. Cette fonction f_X est déterminée à partir de la distribution $\mathcal{G}(A, n, l)$ des graphes générés par A. En outre, nous avons prouvé que e est toujours comprise dans l'intervalle $[0, 2)$.

Étant donné que notre évaluateur se repose sur $\mathcal{G}(A, n, l)$, il convient de fixer n, l'ordre des graphes et l, le nombre de graphes à générer. Comme dit dans la section précédente, l doit être suffisamment élevé pour que le générateur A ait davantage de chance de générer tous les k graphes uniques possédant n sommets. En réalité, l contrôle le niveau de précision et de fiabilité de l'erreur e: si l est trop petit, e sera élevée et surestimée. Pour bien comprendre cela, prenons un cas extrême où $l = 1$ et $n = 8$ ($k = 12\,346$). Dans ce cas, un seul graphe est généré parmi les 12 346. Par conséquent, e sera proche de 2. Toutefois, ce n'est pas pour autant que A est un mauvais générateur de graphes. De même, il est inutile de fixer l à une valeur démesurément grande par rapport à k.

Pour évaluer et comparer les trois générateurs, nous avons décidé de fixer l à $\alpha \times k$, où $\alpha \in \mathbb{N}_{>0}$. Une valeur de $\alpha = 1\,000$ semble être un choix raisonnable; cela offre la possibilité à chaque générateur de générer 1 000 fois tous les k graphes différents d'ordre n. Chaque générateur a ainsi plus chance de générer tous ces graphes au moins une fois. Pour $n = 9$ et $\alpha = 1\,000$, l est ainsi égal à 274 668 000. Sur notre machine[8], cela se traduit par huit heures de calcul pour un seul générateur. De ce fait, pour $n = 10$, nous ne pourrons pas malheureusement évaluer les trois générateurs avec $\alpha = 1\,000$ (cela requerrait un peu moins de six jours de calcul pour chacun). Cependant, avec $\alpha = 50$, nous avons $l = 600\,258\,400$, ce qui représente un temps de calcul plus acceptable. En revanche, l'erreur e sera regrettablement moins fiable pour $n = 10$. En conclusion, pour n allant de 1 à 9, nous optons pour $\alpha = 1\,000$ et, pour $n = 10$, $\alpha = 50$.

Nous pouvons à présent évaluer les générateurs de graphes GNM-Der, GNP-Der et GNM-Norm pour $n \in \{1, 2, \ldots, 10\}$. Le Tableau 5.1 répertorie les erreurs obtenues pour chaque générateur et pour chaque n. Le Diagramme 5.1 montre cette même information, mais de manière plus visuelle. Nous constatons que GNM-Der semble être le plus efficace lorsque $1 \leqslant n \leqslant 4$. Néanmoins, l'erreur e commise par celui-ci augmente de façon importante en fonction de n ($e > 1$ quand $n \geqslant 7$). GNM-Der se révèle donc être un mauvais générateur pour des graphes avec un haut ordre. Quand $5 \leqslant n \leqslant 10$, GNP-Der et GNM-Norm commettent une erreur plus faible que celle entraînée par GNM-Der. De plus, plus n croît et plus l'erreur commise par ces deux générateurs diminue, atteignant une valeur d'approximativement 0,32 pour $n = 10$. GNP-Der et GNM-Norm paraissent donc être de bons générateurs de graphes si les graphes à générer possèdent une quantité assez importante de sommets n. Concernant l'erreur commise par ces deux générateurs, nous nous apercevons que celle provenant de GNM-Norm est plus faible que celle engendrée par GNP-Der pour $1 \leqslant n \leqslant 10$. Nous remarquons malgré tout que l'écart entre ces deux erreurs devient de plus en plus faible au fur et à mesure que n augmente. Tentons d'expliquer cela.

8. Notre machine, un MacBook Pro de 2017, intègre 16 Go de mémoire vive et possède un Intel Core i7-7920HQ comprenant quatre cœurs cadencés à 3,1 GHz.

Ordre n	**Erreur commise par**		
	GNM-Der	**GNP-Der**	**GNM-Norm**
1	0	0	0
2	0,005	0,017	0,008
3	0,016	0,501	0,229
4	0,5127	0,5889	0,5284
5	0,6819	0,6443	0,5802
6	0,9414	0,6685	0,645
7	1,0618	0,6126	0,5794
8	1,1788	0,5114	0,5076
9	1,259	0,4428	0,4323
10	1,3277	0,3214	0,3177

Tableau 5.1 – Erreur commise par chaque générateur en fonction de l'ordre n

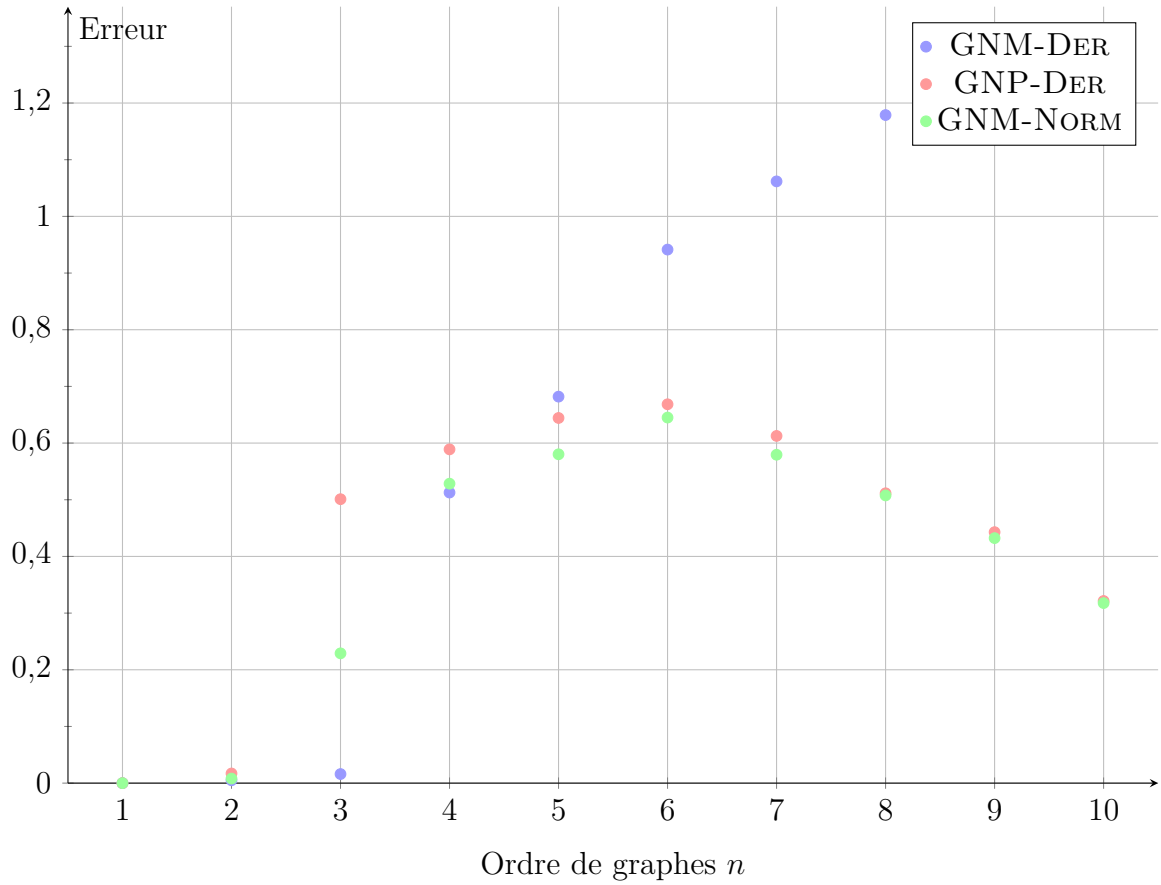


Diagramme 5.1 – Évolution de l'erreur en fonction de l'ordre de graphes n par générateur

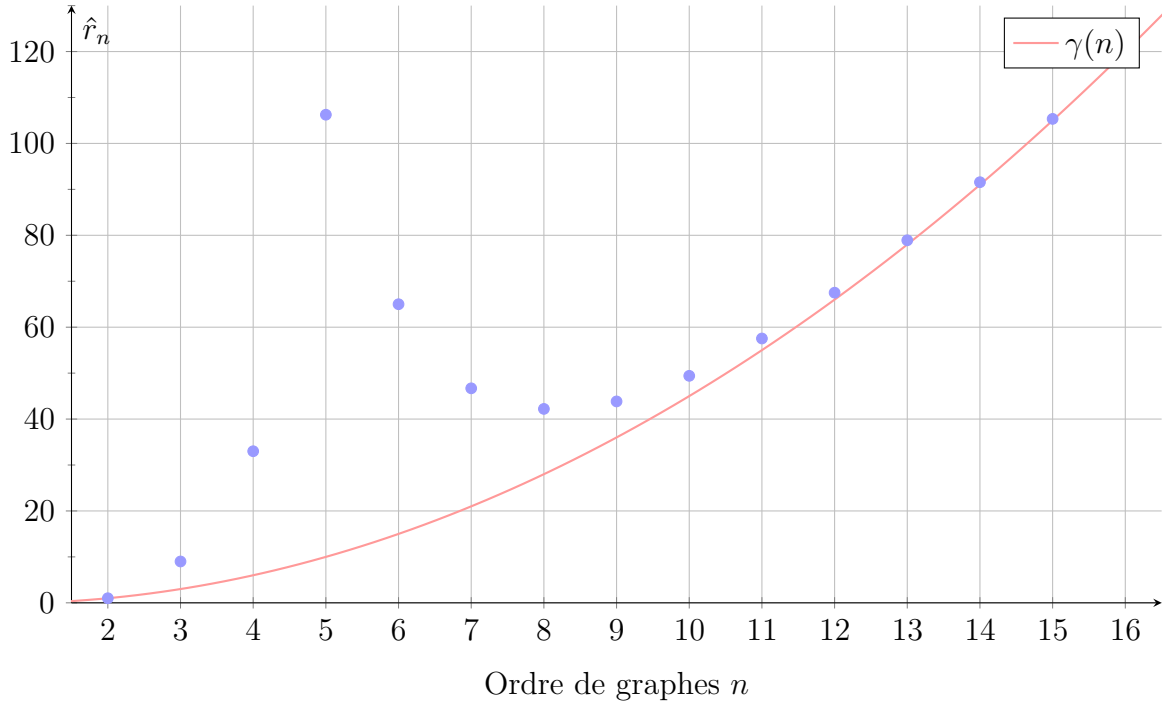


Diagramme 5.2 – Évolution de $\hat{r}_n$ en fonction de l'ordre de graphes n

Revenons au modèle $G(n,p)$ qui est à la base de GNP-Der. Posons $\mathcal{S}_{\text{GNP}}(n,p)$ la distribution de la taille des graphes d'ordre n générés par ce modèle. Il s'avère que $\mathcal{S}_{\text{GNP}}(n,p)$ suit une loi binomiale $\mathcal{B}(r,p)$ avec $r = \gamma(n)$, où $\gamma(n)$ correspond à la taille maximale d'un graphe d'ordre n. En effet, lors de la création d'un graphe avec le modèle $G(n,p)$, chaque arête possible est sélectionnée avec une probabilité égale à p. Dans la Section 3.1, nous avions dit que cette sélection suit une loi de Bernoulli $\mathcal{B}(p)$. Comme nous effectuons $\gamma(n)$ fois cette épreuve de Bernoulli, nous avons que $\mathcal{S}_{\text{GNP}}(n,p)$ suit une loi binomiale $\mathcal{B}(\gamma(n),p)$. Puisque GNP-Der est un cas particulier du modèle $G(n,p)$ où $p = 0{,}5$, la distribution de la taille des graphes générés par GNP-Der, $\mathcal{S}_{\text{GNP}}(n,0{,}5)$, suit la loi binomiale $\mathcal{B}(\gamma(n),0{,}5)$.

Analysons maintenant la loi binomiale analogue à $\mathcal{S}(n)$, notée $\mathcal{R}(n)$, avec $\mathcal{S}(n)$ la distribution de la taille de tous les graphes uniques d'ordre n. Précisément, regardons la tendance des deux paramètres de cette loi en fonction de n. Rappelons que ceux-ci sont estimés par $\hat{r}_n$ et $\hat{p}_n$ *via* les Équations (4.3) et (4.4). Les Diagrammes 5.2 et 5.3 montrent l'évolution de $\hat{r}_n$ et de $\hat{p}_n$ en fonction de n. Nous nous apercevons que plus n devient élevé et plus $\hat{p}_n$ tend vers 0,5 et $\hat{r}_n$ se rapproche progressivement de $\gamma(n)$. En conséquence, cela suggère que lorsque n est grand, $\mathcal{R}(n)$ suit la loi binomiale $\mathcal{B}(\gamma(n),0{,}5)$. C'est également vrai pour $\mathcal{S}(n)$ vu que $\mathcal{R}(n)$ est une très bonne approximation de $\mathcal{S}(n)$.

Nous avons dès lors que $\mathcal{S}_{\text{GNP}}(n,0{,}5)$ et $\mathcal{S}(n)$ suivent la même loi binomiale $\mathcal{B}(\gamma(n),0{,}5)$ pour n grand. En d'autres termes, cela indique que, pour un tel ordre n, la taille des graphes générés par GNP-Der suit la vraie distribution de la taille de tous les graphes différents. Par conséquent, lorsque n est élevé, il est impossible de faire mieux que GNP-Der en ce qui concerne la taille des graphes générés. Quant à GNM-Norm, il devrait donc être moins efficace que GNP-Der étant donné que la distribution de la taille des

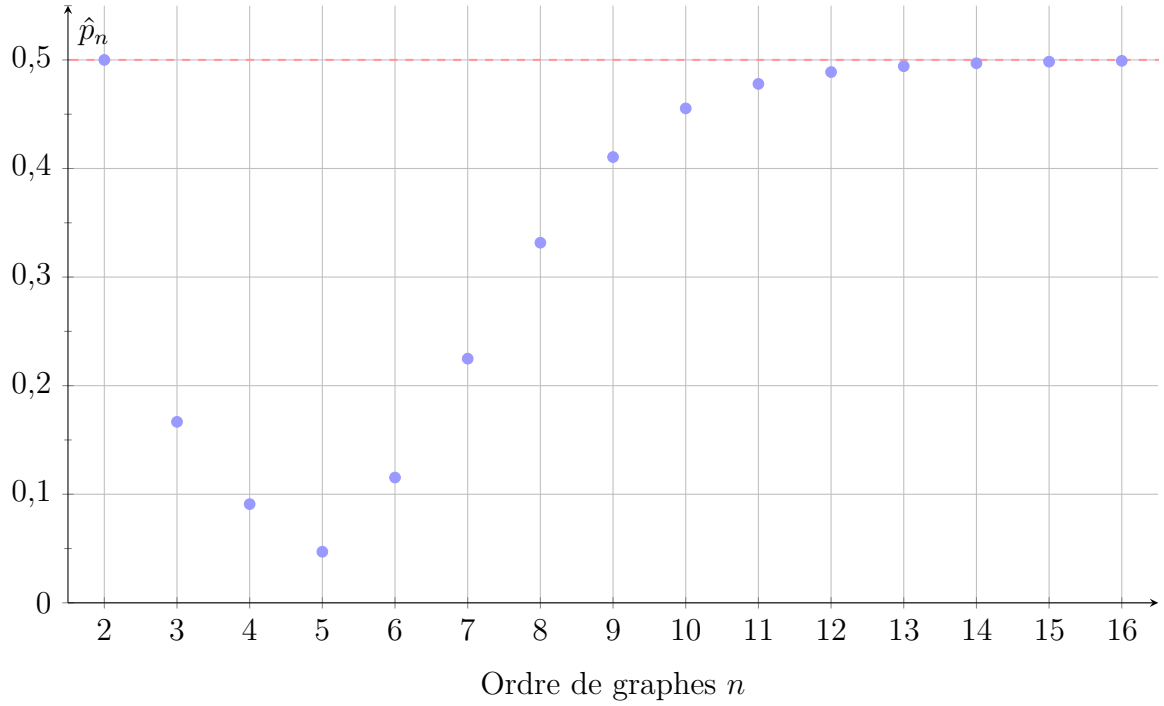


Diagramme 5.3 – Évolution de $\hat{p}_n$ en fonction de l'ordre de graphes n

graphes générés par GNM-Norm n'est qu'une simple approximation de $\mathcal{S}(n)$.

Pour conclure, pour générer des graphes d'ordre n, nous recommandons d'utiliser le générateur GNM-Der si $1 \leqslant n \leqslant 4$, GNM-Norm si $5 \leqslant n \leqslant 10$ et GNP-Der si $11 \leqslant n$. Nous pouvons en outre imaginer un générateur de graphes A hybride distinguant ces cas et appelant le générateur B approprié. Celui-ci déterminerait de ce fait B comme ceci:

$$B \coloneqq \begin{cases} \text{GNM-Der} & \text{si } 1 \leqslant n \leqslant 4, \\ \text{GNM-Norm} & \text{si } 5 \leqslant n \leqslant 10, \\ \text{GNP-Der} & \text{si } 11 \leqslant n. \end{cases}$$

6 Conclusion

Lors de ce travail, nous avons étudié les générateurs de graphes aléatoires. En particulier, nous nous sommes intéressés à la qualité de génération de ceux-ci et avons déclaré que plus les graphes générés par un générateur A s'approchent d'une loi uniforme et plus A est sera efficace. L'objectif était donc de concevoir un évaluateur capable de juger un générateur sur ce critère et de comparer, grâce à celui-ci, deux générateurs A et B. Par la même occasion, nous avons conçu notre propre générateur de graphes. Pour réaliser tout cela, nous avons considéré uniquement les graphes non orientés et avons émis l'hypothèse que deux graphes sont identiques si et seulement s'ils sont isomorphes.

Dans un premier temps, nous avons commencé par décrire le modèle $G(n, m)$ et le modèle $G(n, p)$, deux générateurs de graphes célèbres. Ce premier génère des graphes d'ordre n et de taille m. Pour chacun, un graphe à n sommets et sans arête est créé et m arêtes, choisies aléatoirement dans les $\gamma(n)$ possibles, y sont ajoutées. Concernant le modèle $G(n, p)$, il génère un graphe en sélectionnant toutes les $\gamma(n)$ arêtes possibles avec une probabilité p et en les ajoutant au graphe à n sommets et zéro arête. Pour avoir de meilleurs générateurs, nous avons convenu de faire varier uniformément le paramètre m pour le modèle $G(n, m)$ et de fixer p à 0,5 pour le modèle $G(n, p)$. Il en résulte les deux algorithmes GNM-Der et GNP-Der respectivement.

Dans un second temps, nous nous sommes attardés au problème d'isomorphisme de graphes. Nous avons résolu celui-ci par l'algorithme de labellisation canonique de Brendan Damien McKay implémenté dans le programme nauty. Un tel algorithme calcule la représentation canonique $C(G)$ d'un graphe G de telle manière que G et tout autre graphe G' sont isomorphes si et seulement si $C(G) = C(G')$.

Dès lors, nous avons dans un troisième temps déterminé la distribution des graphes générés par les deux générateurs GNM-Der et GNP-Der pour plusieurs ordres n. Pour ce faire, pour chacun de ceux-ci, nous avons d'abord généré l graphes à n sommets. Ensuite, nous avons calculé la représentation canonique de ceux-ci. Après, nous avons compté toutes les occurrences de chaque représentation canonique. Enfin, nous avons ajouté une occurrence de 0 pour les graphes qui n'ont pas été générés par le générateur parmi tous ceux qui existent. La distribution des graphes générés par un générateur n'est malheureusement pas suffisante pour conclure qu'un générateur A est plus efficace qu'un générateur B.

Nous avons dans un quatrième temps conçu un évaluateur de générateurs de graphes. Pour cela, nous avons défini au préalable le générateur de graphes parfait. La particularité de celui-ci est que la distribution des graphes qu'il génère suit une loi uniforme. À partir d'un générateur de graphes A, notre évaluateur procède en quatre étapes. La première consiste à calculer la fonction de masse f_X de A; cela est réalisé en normalisant la distribution des graphes d'ordre n générés par A. Durant la deuxième étape, la fonction de masse f_Y du générateur de graphes parfait est déterminée. Lors de la troisième étape, pour chaque probabilité de f_X, la distance entre celle-ci et la probabilité correspondante dans f_Y est calculée. Au cours de la dernière étape, la somme de ces distances est réalisée et retournée. Cette somme peut d'ailleurs être vue comme une erreur commise par A. À la fin de la Section 3.4, nous avons prouvé que cette erreur est toujours comprise dans l'intervalle $[0, 2)$.

Dans un cinquième temps, nous avons analysé la distribution de la taille de tous les graphes uniques d'ordre n. Nous avons noté celle-ci $\mathcal{S}(n)$. Pour analyser cette distribution, nous l'avons affichée et observée pour différentes valeurs de n. Celle-ci donnait l'impression de suivre une loi normale ainsi qu'une loi binomiale. Pour confirmer cela, nous avons réalisé le test d'ajustement du khi-deux, avec n allant de 5 à 8, afin de vérifier que $\mathcal{S}(n)$ suit une loi normale et une loi binomiale. Pour cela, nous avons eu recours à l'estimation des paramètres de ces deux lois. Au vu des p-valeurs obtenues, cela suggérait que $\mathcal{S}(n)$ suit bel et bien une loi normale et une loi binomiale. Puisque les p-valeurs étaient plus faibles pour la loi binomiale que les p-valeurs pour la loi normale, nous avons conclu que $\mathcal{S}(n)$ semblait davantage suivre cette seconde loi.

Dans un sixième temps, notre objectif était de trouver une approximation de $\mathcal{S}(n)$ pour tout n. Nous avons donc regardé attentivement l'évolution des paramètres μ_n et σ_n de la loi normale analogue à $\mathcal{S}(n)$ en fonction de n. Il s'est avéré que μ_n peut être déterminé avec exactitude par $\gamma(n)/2$, où $\gamma(n)$ est le nombre maximal d'arêtes d'un graphe d'ordre n. Quant au paramètre σ_n, la relation entre celui-ci et n paraissait plutôt linéaire. Nous avons donc décidé d'effectuer une régression linéaire simple et avons obtenu l'équation suivante:

$$\sigma_n \approx 0{,}3375899521271764n + 0{,}1575828904704868.$$

Nous avons dans un septième temps créé notre générateur de graphes à partir de notre approximation de $\mathcal{S}(n)$. Celui-ci est basé sur le modèle $G(n, m)$ comme GNM-Der. Cependant, contrairement à ce dernier, le nombre d'arêtes m est généré en suivant une loi normale; les paramètres de celle-ci sont estimés par notre approximation de $\mathcal{S}(n)$. Nous avons baptisé ce générateur GNM-Norm.

Dans un dernier temps, nous avons comparé les trois générateurs de graphes GNM-Der, GNP-Der et GNM-Norm à l'aide de notre évaluateur de générateurs. Pour réaliser cela, nous avons évalué l'erreur commise par ces trois générateurs pour $n \in \{1, 2, \ldots, 10\}$. Pour $n \leqslant 4$, GNM-Der a été le plus efficace avec une erreur e inférieure à 0,52. Toutefois, celle-ci augmente en fonction de n de manière conséquente; pour $n \geqslant 7$, e est supérieure à 1. Pour les deux autres générateurs, GNP-Der et GNM-Norm, l'erreur engendrée par GNM-Norm est la plus faible pour $5 \leqslant n \leqslant 10$. Néanmoins, ce générateur n'est distancé que de peu par GNP-Der. Nous avons remarqué que plus la valeur de n augmente et plus l'erreur provenant de ces deux générateurs diminuent; celle-ci est inférieure à 0,35 pour $n = 10$. Également, plus n croît et plus l'erreur commise par GNM-Norm devient de plus en plus proche de celle entraînée par GNP-Der. Pour de grandes valeurs pour n, nous avons en fait découvert que la distribution de la taille des graphes générés par GNP-Der ainsi que la distribution $\mathcal{S}(n)$ suivent la même loi binomiale $\mathcal{B}(\gamma(n), 0{,}5)$. Comme GNM-Norm réalise qu'une modeste approximation de $\mathcal{S}(n)$, nous en avons déduit que GNP-Der est un meilleur générateur que GNM-Norm lorsque n est élevé (supérieur à 10).

Pour conclure, nous avons durant ce travail construit un évaluateur de générateurs de graphes et avons confectionné notre propre générateur basé sur le modèle $G(n, m)$. Pour la génération de graphes d'ordre n avec $5 \leqslant n \leqslant 10$, celui-ci s'est révélé plus efficace que les deux générateurs GNM-Der et GNP-Der, dérivées du modèle $G(n, m)$ et du modèle $G(n, p)$ respectivement. Cependant, nous avons montré que GNP-Der surpasse notre générateur lorsque n est plus élevé.

Bibliographie

[1] Robert Gardner Bartle and Donald R Sherbert. Introduction to real analysis. Wiley, Hoboken, NJ, 2011.

[2] Béla Bollobás. Random Graphs. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2001. URL: https://books.google.be/books?id=o9WecWgilzYC.

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Third Edition. The MIT Press, 3rd edition, 2009.

[4] Reinhard Diestel. Graph Theory: 5th edition. Springer Graduate Texts in Mathematics. Springer-Verlag, © Reinhard Diestel, 2017. URL: https://books.google.be/books?id=zIxRDwAAQBAJ.

[5] P. Erdős and A. Rényi. On random graphs, i. Publicationes Mathematicae (Debrecen), 6:290–297, 1959. URL: http://snap.stanford.edu/class/cs224w-readings/erdos59random.pdf.

[6] P. Erdős and A. Rényi. On the evolution of random graphs. Publication of the Mathematical Institute of the Hungarian Academy of Sciences, pages 17–61, 1960. URL: http://snap.stanford.edu/class/cs224w-readings/erdos60random.pdf.

[7] Tony Finch. Incremental calculation of weighted mean and variance. http://people.ds.cam.ac.uk/fanf2/hermes/doc/antiforgery/stats.pdf, 02 2009.

[8] Scott Fortin. University of alberta: The graph isomorphism problem. https://pdfs.semanticscholar.org/3da3/ef796c6af4bafe2c0d023fc49f9646f4d949.pdf, 1996.

[9] E. N. Gilbert. Random graphs. The Annals of Mathematical Statistics, 30(4):1141–1144, 12 1959. URL: http://snap.stanford.edu/class/cs224w-readings/erdos59random.pdf, doi:10.1214/aoms/1177706098.

[10] Brendan D. McKay. Formats graph6, sparse6 et digraph6. https://users.cecs.anu.edu.au/~bdm/data/formats.html.

[11] Brendan D. McKay. Practical graph isomorphism. Congr. Numer., 1981. URL: https://users.cecs.anu.edu.au/~bdm/nauty/.

[12] Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. CoRR, abs/1301.1493, 2013.

[13] R. Milo, N. Kashtan, S. Itzkovitz, M. E. J. Newman, and U. Alon. On the uniform generation of random graphs with prescribed degree sequences. 2004. URL: https://arxiv.org/abs/cond-mat/0312028v2.

[14] Sadegh Nobari, Xuesong Lu, Panagiotis Karras, and Stéphane Bressan. Fast random graph generation. In Proceedings of the 14th International Conference on Extending Database Technology, EDBT/ICDT '11, pages 331–342, New York, NY, USA, 2011. ACM. URL: http://doi.acm.org/10.1145/1951365.1951406, doi:10.1145/1951365.1951406.

[15] Anne Perrut. Université claude bernard lyon 1: Cours de probabilités et statistiques. http://math.univ-lyon1.fr/irem/IMG/pdf/PolyTunis_A_Perrut.pdf, 2010.

[16] Rajeev Raman. Generating random graphs efficiently. In Frank Dehne, Frantisek Fiala, and Waldemar W. Koczkodaj, editors, Advances in Computing and Information — ICCI '91, pages 149–160, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

[17] Fabien Viger and Matthieu Latapy. Efficient and simple generation of random simple connected graphs with prescribed degree sequence. In Lusheng Wang, editor, Computing and Combinatorics, pages 440–449, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[18] Fabien Viger and Matthieu Latapy. Fast generation of random connected graphs with prescribed degrees. CoRR, 2005.

[19] Christian Walck. Hand-book on statistical distributions for experimentalists. http://www.stat.rice.edu/~dobelman/textfiles/DistributionsHandbook.pdf, 2007.

[20] Larry Wasserman. All of Statistics: A Concise Course in Statistical Inference. Springer Publishing Company, Incorporated, 2010.

A Format graph6

Le format graph6 [10] est une technique de représentation de graphes. Il a pour but de représenter un graphe non orienté d'ordre inférieur à $2^{36}-1$ en une chaîne de caractères. Dans cette annexe, nous allons présenter une méthode permettant de calculer le format graph6 d'un graphe à partir de sa matrice d'adjacence.

Pour ce faire, définissons deux fonctions $R(\vec{x})$ et $N(n)$. Commençons par la première, $R(\vec{x})$. Elle reçoit en entrée un vecteur binaire $\vec{x}$ de taille $|\vec{x}|$ et retourne une suite d'octets. Elle traite $\vec{x}$ comme ceci:

1. créer $\vec{y}$ en ajoutant plusieurs 0 à droite de $\vec{x}$ de manière à ce que $|\vec{y}|$ devienne un multiple de 6, c'est-à-dire tant que $|\vec{y}| \mod 6 \neq 0$;
2. scinder $\vec{y}$ en vecteurs $\vec{y}_i$ de 6 bits ($|\vec{y}|/6$ vecteurs sont créés);
3. ajouter 63 à chaque $\vec{y}_i$;
4. retourne chaque vecteur $\vec{y}_i$ sous forme d'octets.

Quant à la fonction $N(n)$, elle reçoit en entrée un naturel $n < 2^{36}-1$ et retourne plusieurs octets. Elle procède de cette façon:

— si $n \leqslant 62$, retourner $n+63$;

— si $62 < n \leqslant 258\,047$, retourner $126, R(\vec{x})$, où $\vec{x}$ est la représentation binaire en *big-endian* de n;

— si $258\,047 < n \leqslant 2^{36}-1$, retourner $126, 126, R(\vec{x})$, où $\vec{x}$ est la représentation binaire en *big-endian* de n.

Soient un graphe non orienté G dont l'ordre est inférieur à $2^{36}-1$ et sa matrice d'adjacence $A = (a_{ij})$. Construisons un vecteur binaire $\vec{t}$ de la façon suivante:

$$\vec{t} = \left[a_{12}, a_{13}, a_{23}, a_{14}, a_{34}, \ldots, a_{(|G|-1)|G|}\right]^{\mathsf{T}}.$$

Ce vecteur contient uniquement le triangle supérieur de A. Comme nous l'avions vu dans la Section 2.1, ce triangle contient toutes les informations nécessaires sur G. Nous avons désormais tous les éléments pour calculer le format graph6 de G. Ce format est défini comme $N(|G|), R(\vec{t})$ auquel nous convertissons chaque octet retourné en le caractère correspondant.

Exemple A.1. Reprenons le graphe non orienté G de la Figure 2.1 présenté dans la Section 2.1. Sa matrice d'adjacence A peut être trouvée à la Page 9. À partir de cette dernière, nous pouvons facilement calculer le vecteur binaire $\vec{t}$:

$$\vec{t} = [1,0,1,0,1,1,0,0,1,1,0,0,0,1,1]^{\mathsf{T}}.$$

Calculons $R(\vec{t})$. Tout d'abord, nous formons $\vec{y}$ en ajoutant quelques 0 à droite de $\vec{t}$ tant que $|\vec{y}|$ n'est pas un multiple de 6. Nous obtenons donc:

$$\vec{y} = [1,0,1,0,1,1,0,0,1,1,0,0,0,1,1,0,0,0]^{\mathsf{T}},$$

avec $|\vec{y}| = 18$. Divisons maintenant $\vec{y}$ en vecteurs $\vec{y}_i$ de 6 bits. Nous en avons trois:

$$\vec{y}_1 = [1,0,1,0,1,1]^{\mathsf{T}},$$

$$\vec{y}_2 = [0, 0, 1, 1, 0, 0]^\mathsf{T},$$
$$\vec{y}_3 = [0, 1, 1, 0, 0, 0]^\mathsf{T}.$$

Ajoutons 63 à ces derniers et convertissons-les en décimal. Cela nous donne: $106, 75, 87$. Il ne reste plus qu'à déterminer $N(|G|)$. Comme $|G| \leqslant 62$, $N(|G'|) = 6 + 63 = 69$. En conséquence, nous avons la suite d'octets suivante: $69, 106, 75, 87$. La chaîne de caractères résultante est donc la suivante: `EjKW`.