Maintenance et simulation de graphes aléatoires dynamiques

Romaric Duvignau

université de BORDEAUX

Thèse présentée
pour obtenir le grade de

Docteur de l'Université de Bordeaux

École Doctorale de Mathématiques et d'Informatique
Spécialité Informatique

Par Romaric Duvignau

Maintenance et simulation de graphes aléatoires dynamiques

Sous la direction de : Philippe Duchon

Soutenue le 16/10/2015

Membres du jury :

M. Duchon Philippe	Professeur (LaBRI)	Directeur de thèse
Mme Gardy Danièle	Professeur (PRISM)	Examinatrice
M. Godard Emmanuel	Professeur (LIF)	Examinateur
M. Klasing Ralf	Directeur de recherche (CNRS, LaBRI)	Président du jury
M. Martínez Conrado	Professeur (Univ. Politècnica de Catalunya)	Rapporteur
M. Nicaud Cyril	Professeur (LIGM)	Examinateur
M. Ravelomanana Vlady	Professeur (LIAFA)	Rapporteur

Titre Maintenance et simulation de graphes aléatoires dynamiques

Résumé Nous étudions le problème de maintenir une distribution donnée de graphes aléatoires après une séquence arbitraire d'insertions et de suppressions de sommets. Dans l'objectif de modéliser l'évolution de réseaux logiques dynamiques, nous travaillons dans un modèle local où l'accès à la liste des sommets est restreint. À la place, nous faisons l'hypothèse d'un accès à une primitive globale qui retourne un sommet aléatoire, choisi uniformément dans l'ensemble total des sommets. Le problème de maintenance a été exploré sur plusieurs modèles simples de graphes aléatoires (graphes d'Erdős–Rényi, graphes basés sur le modèle par paires, graphes k-sortants uniformes). Pour chacun des modèles, un ou plusieurs algorithmes pour la tâche de maintenance ont été décris et analysés ; les plus élaborés de ces algorithmes sont asymptotiquement optimaux. Le problème de maintenance soulève plusieurs problèmes de simulation liés à notre contexte distribué. Nous nous sommes intéressé en particulier à la maintenabilité de distributions de graphes et à la simulabilité de familles de distributions de probabilité sur les entiers, dans le modèle d'aléa présenté. Une attention particulière a été portée sur la simulation efficace de lois spécifiques nous intéressant (certaines lois binomiales). Cette dernière a pu être obtenue en exploitant les propriétés d'un nouvel arbre de génération pour les permutations, que nous avons introduit.

Mots-clés graphes aléatoires, graphes dynamiques, maintenance de réseaux logiques, préservation d'aléa, génération aléatoire, simulabilité

Title Maintenance and simulation of dynamic random graphs

Abstract We study the problem of maintaining a given distribution of random graphs under an arbitrary sequence of vertex insertions and deletions. Keeping in mind our objective to model the evolution of dynamic logical networks, we work in a local model where we do not have direct access to the list of all vertices. Instead, we assume access to a global primitive that returns a random vertex, chosen uniformly from the whole vertex set. The maintenance problem has been explored on several simple random graph models (Erdős–Rényi random graphs, pairing model based random graphs, uniform k-out graphs). For each model, one or several update algorithms for the maintenance task have been described and analyzed ; the most elaborate of them are asymptically optimal. The maintenance task rise several simulation issues linked to our distributed context. In particular, we have focused on maintenability of random graph distributions and simulability of families of probability distributions over integers in our local random model. Special attention has been paid to efficient simulation of particular distributions we were interested in (certain binomial distributions). The latter has been obtained through the use of properties of a new generation tree for permutations, which has been introduced along the way.

Keywords random graphs, dynamic graphs, logical network maintenance, randomness preservation, random generation, simulability

---

Laboratoire Bordelais de Recherche en Informatique - UMR 5800

Remerciements

Comme il est désormais de tradition, et que j'apprécie particulièrement les traditions, j'aimerais remercier ici l'ensemble des personnes qui ont participé de manière plus ou moins directe à la publication de cette thèse.

En premier lieu, je tiens à remercier singulièrement mon directeur de thèse Philippe Duchon. Je le remercie en outre de ses très bons conseils et de son excellent encadrement qu'il a su me prodiguer au cours de ces trois années de doctorat. De par sa personnalité, ce fut de plus toujours un plaisir de travailler à ses côtés.

J'aimerais aussi remercier chaleureusement mes deux relecteurs Conrado Martínez et Vlady Ravelomanana d'avoir accepté de prendre le temps de relire ma thèse, ainsi que pour leur rapport détaillé et encourageant.

Je remercie également Ralf Klasing qui a gentiment accepté de présider mon jury *en français*.

Danièle Gardy, Emmanuel Godard et Cyril Nicaud m'ont fait l'honneur de faire partie de mon jury de thèse, et je les en remercie pour cela.

Je remercie également

— Tous les membres du LaBRI avec lesquels j'ai pu interagir, par exemple, sans essayer d'être exhaustif, je me souviens de quelques discussion techniques enrichissantes avec Pierre Castéran, Géraud Sénizergues, Paul Dorbec, Olivier Guibert, Jean-Christophe Aval et Nicolas Bonichon (pris dans un ordre aléatoire). De plus, je tiens à remercier spécifiquement les chercheurs participants aux groupes de travail Algorithmique Distribuée, Combinatoire Énumérative et Algébrique et Graphes et Applications, pour animer trois groupes de travail vivants et scientifiquement très intéressants. Je me dois aussi de remercier l'ensemble des doctorants pour faire perdurer la bonne ambiance qui règne parmi les thésards et enfin l'AFoDIB pour l'organisation d'événements afin de se voir, aussi, en dehors du laboratoire.

— Tous les membres du groupe de travail Aléa (et Alea in Europe), pour organiser et participer chaque année aux journées aléa, proposant un format de rencontre atypique à l'esprit singulier, alternant exposés scientifiques, soirées jeux et bouillabaisse.

— Pour finir, j'ai une pensée particulière envers mes co-bureaux, présents et anciens : Thao, Tung, David, Joris, Claire, Lorijn, Justine, et François (par ordre de longueur, puis lexicographique), avec qui nous avons toujours trouvé des sujets de conversation pour faire diverger, de temps en temps, le nez de

l'écran. J'aurais enfin une pensée, pour mon ami et collègue Noël Gillet, qui a du me supporter depuis la première année d'université, et qui s'est proposé d'organiser mon pot de thèse (une proposition, avouons-le, que l'on ne peut refuser).

Table des matières

Introduction

Cette thèse explore la maintenance de distributions de graphes aléatoires dans un contexte décentralisé. Ce travail étend le domaine d'étude de la conservation d'aléa dans deux directions : en travaillant sur une structure de données plus générale que celles étudiées jusque-là, et en se plaçant dans un modèle décentralisé, où l'accès à la liste des sommets du graphe n'est pas autorisé.

La problématique de *préservation d'aléa* tient sûrement pour origine un article de Knuth de 1977, *Deletions that preserve randomness* [65]. Il était déjà connu à l'époque (voir [59]) qu'un arbre binaire de recherche, ou ABR, obtenu après l'insertion de n clés distinctes dans un ordre uniforme, suivi d'une unique suppression d'une des clés choisie uniformément, résulte en un arbre binaire dont la forme suit la même distribution qu'un ABR aléatoire [1] (dans la terminologie actuelle).

Knuth montre [2] que d'une manière un peu contre-intuitive, la propriété n'est plus vérifiée lorsque plusieurs suppressions sont effectuées successivement ; dans cet article, plusieurs modèles pour l'insertion de nouvelles clés, et la suppression de clés déjà présentes sont proposés et analysés vis-à-vis de leur *insensibilité* à la suppression. Cette insensibilité est vérifiée si la structure aléatoire obtenue après l'ensemble des modifications suit la même distribution que si elle avait été produite par l'insertion des clés restantes dans un ordre uniforme. Ici, l'*aléa* porte sur la façon de choisir la prochaine clé à insérer ou à supprimer, mais la structure de données reste entièrement déterministe.

L'idée qu'il est souhaitable de garantir la distribution de l'objet aléatoire obtenu après une série d'insertions et de suppressions a conduit plusieurs auteurs à *déplacer* l'aléa portant sur l'opération de mise à jour (ajout/suppression d'éléments) vers les algorithmes opérant sur la structure ou la structure elle-même en la modifiant légèrement (voir [75]). Cette méthode permet de briser toute dépendance entre la structure finale et la séquence des mises à jour : le choix des opérations et des éléments ajoutés/supprimés peuvent être laissés à un adversaire, la distribution de la structure de données est garantie pour toute séquence de mises à jour.

Pugh introduit en 1990 [82] un des premiers modèles de structure de données qui présente des caractéristiques de conservation d'aléa. Le modèle présenté, les *skip-list* ou en français *listes à enjambements*, est une alternative probabiliste aux

1. ABR généré en insérant les clés présentes dans un ordre aléatoire uniforme.
2. Précisément, ce phénomène est mentionné pour la première fois dans la thèse de Knott [64].

arbres équilibrés. Une telle liste est composée d'un ensemble de listes chaînées triées appelées niveaux ; chaque niveau contient un sous-ensemble des clés (le premier les contient toutes), et chaque clé présente au niveau i est présente au niveau $i+1$ avec probabilité fixe p (un paramètre du modèle), et avec probabilité complémentaire $1-p$, elle n'apparaîtra pas aux niveaux supérieurs.

Lorsque la structure contient n clés, le temps de recherche d'une clé est en $\mathcal{O}(\log(n))$ avec forte probabilité alors qu'il est de $\mathcal{O}(\log(n))$ dans le pire des cas pour les arbres équilibrés. En revanche, les algorithmes de *maintenance* sont significativement plus simples que pour les arbres équilibrés et la structure ne présente pas de dégénérescence (c'est-à-dire temps de recherche linéaire) pour certaines séquences d'insertions, un défaut que possèdent les arbres de recherche classiques.

De manière indépendante et simultanément à Pugh, Aragon et Seidel [86] introduisent un modèle d'arbre binaire aléatoire offrant des performances similaires aux listes à enjambements de Pugh. Dans les arbretas ou *treap*, chaque clé est associée à une *priorité*. La structure de données correspond à l'ABR obtenu lorsque les clés sont ajoutées via l'algorithme d'insertion classique dans un ABR selon l'ordre des priorités ; l'arbre obtenu respecte alors la propriété de tas sur les priorités. Cette propriété est conservée après chaque mise à jour de la structure. Lorsque les priorités sont indépendantes et choisies uniformément dans $[0,1]$, l'arbretas aléatoire correspond à un ABR aléatoire (pour les clés), et les bonnes performances des opérations usuelles, e.g. recherche/insertion/suppression, sont ainsi récupérées.

Le modèle présenté peut aussi s'adapter au cas où la recherche de certains éléments est plus fréquente que d'autres. En pratique, leur performance se compare très favorablement par rapport aux arbres binaires de recherche équilibrés (arbre bicolore, arbre aa), aux arbres radix et aux listes à enjambements (voir e.g. [57]).

Un autre modèle d'arbres aléatoires, les arbres binaires de recherche randomisés sont introduits par Martínez et Roura [73] en 1998. Les auteurs présentent des algorithmes d'insertion et de suppression *probabilistes* pour les ABR, qui possèdent la propriété de préserver la distribution des ABR aléatoires dont un certain nombre de propriétés a déjà été étudiées (voir e.g. [85]). Plus précisément, leurs algorithmes garantissent qu'une insertion ou une suppression dans un ABR aléatoire, produit un ABR aléatoire sur le nouvel ensemble de clés. Les algorithmes sont simples à implémenter et nécessitent uniquement de faire des tirages aléatoires de nombres entiers entre 0 et la taille du sous-arbre courant ; pour des questions de performance, cette taille est stockée dans chaque nœud de l'ABR. Ce modèle possède l'avantage de ne pas faire intervenir des nombres réels contrairement aux arbretas, même si en pratique dans ces derniers, il suffit de stocker seulement quelques bits par priorité en effectuant des comparaisons paresseuses.

La différence majeure entre le modèle de Martínez *et al.* et les arbretas, est d'ordre conceptuelle : un arbre binaire de recherche randomisé est un arbre binaire de recherche, dont l'information supplémentaire contenue dans les nœuds est *purement redondante* et est présente uniquement pour obtenir de bonnes performances ; elle pourrait très bien être recalculée à chaque opération de mise à jour. Dans les

arbretas, l'information supplémentaire stockée sur chaque nœud (la priorité) fait partie intégrante de la structure de données, qui ne correspond plus à proprement parler uniquement à un arbre binaire de recherche.

Les ABR randomisés ont été depuis *généralisés* à des structures de données multi-dimensionnelles. Un arbre k-d est une structure de données partitionnant un espace k-dimensionnel afin d'offrir un bon compromis temps/espace pour résoudre des opérations de type recherche de clés, et de mise à jour de la structure (voir [13]). L'insertion, la suppression et la recherche d'une clé prend un temps logarithmique dans un arbre k-d contruit aléatoirement, obtenu par exemple lorsque les clés successivement insérées sont des k-uplets uniformes de $[0,1]^k$. Dans l'article [35] de Duch *et al.*, le modèle d'arbres k-d randomisés est introduit et permet de garantir des performances logarithmiques en moyenne sur les opérations de base quelque soit la séquence des mises à jour. Cette garantie est obtenue en préservant exactement la distribution des arbres k-d randomisés. Le modèle présente ainsi une généralisation du modèle d'ABR randomisé à plusieurs dimensions. Une autre généralisation introduite par Duch [34] est le modèle de *quadtree* aléatoires. Un *quadtree* est un autre type de structure de données de type arbre, partitionnant un espace multidimensionnel (généralement bidimensionnel). La distribution des *quadtree* aléatoires (obtenus avec un ordre d'insertion des clés uniforme) est maintenue exactement dans [34] après chaque insertion et suppression ; cette maintenance permet de remplacer l'algorithme de suppression standard, complexe à adapter aux dimensions supérieures et induisant une dégénérescence de l'arbre (augmentation du coût moyen de recherche d'un nœud au cours de suppressions successives).

À présent lorsque nous parlerons de *préservation d'aléa*, il faut bien comprendre la notion utilisée par Martínez *et al.* et dont le modèle des arbres binaires de recherche randomisés est le premier à inclure un algorithme de suppression *préservant* la distribution des ABR aléatoires. L'aléatoire est *introduit* par le biais d'algorithmes probabilistes et aucune hypothèse n'est formulée quant à la séquence de mises à jour. Une notion différente de préservation [3] a été étudiée dans [58], où un algorithme de suppression déterministe est introduit pour les ABR ordonnés (variante d'ABR dans lequel les dates d'insertions sont conservées). L'algorithme présenté permet d'obtenir ainsi après une suppression, exactement la même structure de données que celle construite lorsque la clé n'a jamais été insérée. Les analyses sont effectuées en supposant que les clés sont insérées dans un ordre uniforme.

Des modèles de graphes aléatoires proposés dans la littérature comme les *skip-graphs* [6] reprennent déjà les idées de modèles à préservation d'aléa. Un tel graphe consiste en une structure de données distribuée où chaque nœud est une ressource (à la place d'une machine) d'un réseau décentralisé. Les nœuds sont connectés entre eux à la manière des listes à enjambements de Pugh, mais chaque niveau contient (potentiellement) plusieurs listes doublement chaînées. Dans un skip-graph, l'opération de base est la recherche de clé qui est effectuée en temps logarithmique avec

3. Cette notion est d'une certaine manière plus proche de l'idée originale de Knuth.

forte probabilité et peut être initialisée à partir de n'importe quel nœud. Le modèle est maintenu au fur et à mesure des mises à jour du réseau.

Nous proposons d'étendre cette notion de *préservation de lois* aux graphes aléatoires *décentralisés* afin de répondre à des problématiques liées à la maintenance de réseaux logiques dynamiques à grande échelle, dont les réseaux *pair à pair* sont les plus représentatifs.

Un réseau *pair à pair*, aussi connu sous le sigle P2P, est un réseau décentralisé dynamique conçu pour partager des données ou des ressources entre différents nœuds (appelés alors *pairs* ou *participants*) du réseau. Le réseau est par nature dynamique, dans le sens où les nœuds participants rejoignent et quittent le réseau de manière ininterrompue et non-prédictible. Des algorithmes *locaux* (n'utilisant pas d'information globale) forment le protocole du réseau et sont exécutés par les nœuds afin de maintenir certaines propriétés désirées. Idéalement, ces algorithmes locaux sont simples et robustes, dans le sens où ils réagissent bien à des séquences de départs et d'arrivées disparates, et potentiellement malicieuses.

Les propriétés que les concepteurs de réseaux recherchent en général incluent une forte connectivité, un faible degré et un diamètre faible. Les caractéristiques particulièrement souhaitées dans le cas des réseaux P2P incluent aussi la décentralisation, l'évolutivité, la tolérance aux pannes, l'équilibrage de charge, et la disponibilité de requêtes complexes et efficaces (incorporant par exemple des critères géographiques ou temporels).

Les premières implémentations de réseaux de type P2P apparaissant au début des années 2000 sont Napster [92], Gnutella [53], Freenet [26] et GNUnet [55] (voir l'étude [1] pour leurs descriptions).

De manière un peu surprenante, les premiers protocoles P2P manquaient indéniablement d'un certain nombre de propriétés recherchées que nous avons citées. Par exemple, nous ne retrouvons pas le passage à l'échelle sur le réseau historique de Napster à cause de l'utilisation d'un serveur central [4] (goulot d'étranglement) et sur Gnutella à cause de requêtes par inondation du réseau (forte complexité en messages). Quant au réseau Freenet, il manque de garantie tant au niveau du temps d'exécution des recherches et de la persistance des données.

Le réseau pair à pair BitTorrent [27] introduit plus récemment présente une architecture mixte dont la partie essentielle (requêtes, partage de ressources) est de type pair à pair. Il est sans aucun doute le réseau P2P le plus populaire au monde [81] et un des plus gros consommateurs de bande passante internet actuellement.

La partie centralisée du protocole se base sur un logiciel appelé *tracker*. Dans un réseau BitTorrent, un pair souhaitant télécharger un fichier amorce ce téléchargement en se connectant au tracker de ce fichier qu'il a pu récupérer via des portails webs dédiés à la recherche de fichiers torrents. L'utilisateur récupère alors du tracker un ensemble aléatoire de pairs possédant le fichier ; une connexion est établie à chacun de ces pairs pour identifier les morceaux du fichier qu'ils détiennent. La phase

4. Un autre protocole populaire, eDonkey2000, a lui aussi disparu potentiellement à cause d'une trop forte centralisation.

de téléchargement à proprement parler commence alors : les morceaux manquants du fichier recherché sont obtenus de ces pairs voisins. Dans le même temps, chaque participant téléverse ou *upload* au maximum à 4 (par défaut) autres pairs.

Une caractéristique propre de BitTorrent est de distinguer deux types de nœuds : les *téléchargeurs* et les *semeurs*. Les premiers sont en phase active de téléchargement et recherchent les fragments qui leurs manquent ; les seconds ont déjà terminé le téléchargement complet du fichier, et restent dans le système uniquement pour servir de sources à des téléchargeurs.

Les graphes qui modélisent les réseaux P2P sont typiquement simples (sans arêtes multiples ni boucles) et non-orientés (exception faite de BitTorrent où une orientation naturelle existe, suivant le sens du téléchargement). Ceux que nous considérerons dans ce travail pourront toutefois être munis d'une orientation, voire de boucles ou d'une multiplicité sur les arcs. En effet, ces éléments du graphe même s'ils ne correspondent pas à des liens logiques dans le réseau peuvent nous permettre de décrire précisément la distribution du réseau à chaque instant et pourront permettre ainsi une maintenance avec de meilleures performances.

Dans la totalité des réseaux cités, il existe une dépendance importante entre la structure du réseau et la séquence de mises à jour. Cette dépendance est nuisible et peut être utilisée afin de construire des stratégies adversariales visant à déstructurer le réseau (voir l'étude comparative [91]). Cette dépendance est à l'origine de plusieurs phénomènes (dérives du système, ajouts d'hypothèses probabilistes sur la dynamicité, difficulté d'analyses) que nous proposons d'atténuer ou d'annuler en se plaçant dans un objectif de maintenance exacte de distributions de graphes aléatoires.

L'étude de l'évolution des réseaux de type P2P a été réalisée jusqu'à présent en faisant des hypothèses fortes quant à la séquence de mises à jour. Dans les analyses apparaissant dans la littérature, le processus d'arrivée de nouveaux nœuds et des départs des nœuds déjà présents est modélisé par un processus de Poisson, *i.e.* un processus probabiliste, sans mémoire et en temps continu.

Par exemple dans [76], les auteurs proposent un protocole pour construire un réseau décentralisé de faible diamètre. L'architecture se base sur un serveur central qui permet d'ajouter un nouveau nœud au réseau et à réparer les connexions perdues. L'évolution du réseau est modélisée par un processus de Poisson de paramètre λ, et les départs (ou temps de présence sur le réseau) sont des exponentielles indépendantes de paramètre μ. Sous ces hypothèses, le protocole proposé maintient avec forte probabilité un graphe dont le diamètre est logarithmique dans le nombre de nœuds présents.

Chord [89] est un réseau de type pair à pair proposant une topologie en anneau et implémentant une table de hachage distribuée. Dans l'analyse [70] qui est faite de l'évolution du réseau, la dynamicité est elle aussi modélisée sous forme probabiliste. La modélisation retenue est la même que pour [76] : les arrivées sont supposées suivre un processus de Poisson de paramètre fixe, et les départs suivent une loi exponentielle elle aussi de paramètre fixe. Sous ces hypothèses, l'évolution du réseau est analysée et il apparaît que le protocole réagit bien à la dynamicité du réseau

(connexe avec forte probabilité, temps faible pour revenir dans un état consistant, etc).

Dans [83], les auteurs cherchent à modéliser plusieurs aspects du protocole BitTorrent afin d'extraire différentes statistiques ; au vu de la complexité du système, sa modélisation est par nature difficile. L'évolutivité est là encore modélisée par un processus de Poisson de paramètre λ pour les nouvelles requêtes, et des départs suivant une loi exponentielle de paramètre fixé à l'avance (pour les deux types de nœuds présents dans le système). D'autres paramètres utilisés incluent la bande passante montante et descendante (considérée identique pour tous les participants). Sous leur modèle, les auteurs fournissent des expressions de plusieurs quantités en moyenne (e.g. nombres de pairs téléchargeurs/semeurs, temps de téléchargement) en fonction des différents paramètres du système.

Cooper *et al.* [28] proposent une modélisation plus générale pour le processus d'arrivées et de départs dans leur analyse du réseau P2P SWAN [23]. Ce réseau se base sur un protocole entièrement décentralisé et construit de nouvelles connexions en se basant sur des marches aléatoires dans le réseau. Dans l'analyse [28] de l'évolution du réseau, les taux de nouvelles arrivées et de départs dépendent de la taille du réseau, contrairement aux taux fixes utilisés dans [76]. Plus précisément, les temps entre deux arrivées sont supposés indépendants et distribués exponentiellement avec espérance ν_n lorsque le réseau contient n nœuds. Les temps de présence des différents nœuds sont aussi supposés indépendants et distribués exponentiellement avec moyenne μ_n. Ainsi, la taille du réseau est modélisée par une chaîne de Markov d'espace d'états $\mathbb{N}$, et de probabilité de transition

$$p_{n,n+1} = \frac{1/\nu_n}{1/\nu_n + n/\mu_n} = \frac{\mu_n}{\mu_n + n\nu_n}, \quad p_{n,n-1} = 1 - p_{n,n+1} = \frac{n\nu_n}{\mu_n + n\nu_n}.$$

En considérant $\nu_n = \nu$ et $\mu_n = \mu$ pour tout n, le processus de [76] est retrouvé.

Depuis ces analyses, il est apparu que modéliser l'évolution des réseaux P2P par un tel processus d'arrivée/départ n'est pas réaliste vis-à-vis des réseaux P2P actuels. Par exemple, dans [80], les auteurs montrent à partir de mesures effectuées sur le réseau BitTorrent que la modélisation des arrivées Poisson et départs exponentiels (utilisé e.g. dans [83]) est en totale contradiction avec les mesures effectuées. D'une certaine manière, le fait que cette modélisation échoue à être fidèle à la réalité justifie la recherche d'algorithmes de mise à jour préservant les bonnes propriétés du réseau sans formuler ces hypothèses sur la dynamicité.

Une autre motivation de cette thèse vient de la difficulté d'analyser l'évolution des réseaux logiques, et en particulier lorsque l'évolution est modélisée de manière non probabiliste. Dans le réseau pair à pair étudié par Cooper, Klasing et Radzik dans [29], un diamètre logarithmique est maintenu avec forte probabilité après chaque mise à jour à l'aide de marches aléatoires. Les bonnes propriétés du réseau (diamètre, connectivité, degrés faibles des sommets) sont maintenues avec forte probabilité, et ce même après des suppressions de sommets et d'arêtes contrôlées par un adversaire. Les modèles d'évolution utilisés sont simples (arrivées uniquement,

et *fifo* : chaque arrivée entraîne le départ du sommet le plus ancien dans le réseau) mais les auteurs concèdent déjà que l'analyse du réseau reste quand même beaucoup plus complexe que dans le cas de graphes aléatoires statiques.

Pour répondre à ces différentes problématiques, nous proposons d'étendre la thématique de préservation d'aléa aux graphes décentralisés et dynamiques. Notre objectif portera alors sur la conception d'algorithmes probabilistes afin de maintenir une distribution de graphes aléatoires donnée après toute séquence de mises à jour, et ce dans un modèle à information *locale*. Les choix effectués dans la conception de ce modèle ont été guidés afin de nous permettre une préservation exacte de loi et d'être applicable aux réseaux pair à pair (de manière plus générale aux réseaux logiques à grande échelle).

Dans notre modèle de localité, étant donné que notre objectif est de modéliser des réseaux logiques dynamiques, nous ferons l'hypothèse que l'accès à l'ensemble du graphe n'est possible qu'au travers d'une seule primitive globale ; la taille du graphe est soit supposée connue des nœuds lors des mises à jour, soit elle est supposée impossible à déterminer (exception faite du cas où la distribution de graphes aléatoires considérée ne comprend que des graphes connexes).

Les différents nœuds du réseau ne peuvent communiquer qu'avec d'autres nœuds dont ils ont appris l'existence. Au lancement d'une mise à jour du graphe, cet ensemble correspond à leur voisinage dans le graphe. Chaque nœud peut ensuite (potentiellement) apprendre l'existence d'un autre nœud en appelant la primitive globale `RandomVertex()`, qui renvoie un sommet uniforme du graphe courant. Lors d'une mise à jour, chaque sommet peut décider d'ajouter et d'enlever des liens logiques (arêtes du graphe) avec les sommets dont il a connaissance. Distincte et indépendamment de la source d'aléa global, *i.e.* de la primitive `RandomVertex()`, chaque nœud est supposé posséder une source d'aléa interne.

Notre primitive globale doit être vue comme une idéalisation du mécanisme intrinsèquement *externe* à tout réseau décentralisé servant à contacter un premier nœud. Ce modèle se raffine en deux sous-modèles suivant si les nœuds partagent la connaissance de la taille du réseau au moment des mises à jour.

Nous verrons dans ce travail, que cette connaissance peut permettre de diminuer la charge d'aléa global nécessaire à la maintenance de graphes aléatoires. Néanmoins, nous montrons qu'il est aussi possible, de maintenir efficacement des distributions intéressantes de graphes aléatoires, et ce sans connaître le nombre de nœuds constituant le réseau lors des différentes mises à jour.

De plus, même sous l'hypothèse d'accès à une primitive d'aléa globale *forte* comme dans notre modèle de décentralisation, il apparaît que des distributions de graphes simples et classiques telle que les graphes d'Erdős–Rényi de densité constante ne sont pas maintenables lorsque la taille du réseau n'est pas accessible.

Notre modèle, bien qu'idéaliste d'un certain point de vue, fait apparaître la difficulté d'allier préservation de loi et vision purement **locale** des nœuds. Cette difficulté peut être insurmontable (graphes d'Erdős–Rényi), entraîner un surcoût non-négligeable lors de mises à jour (différents modèles par paire), ou être surmontée

au prix d'algorithmes plus complexes mais d'une surcharge *relativement faible* sur l'aléa global (graphe k-sortants).

Nous présentons à présent le plan de cette thèse qui se divise en cinq chapitres.

Le chapitre 1 introduit formellement ce que nous entendons par maintenance de distributions de graphes aléatoires. Nous présentons une définition très générale de la tâche de maintenance qui peut s'appliquer à d'autres objets que les graphes aléatoires. Ce chapitre comprend ensuite une argumentation sur cette définition et nous montrons que les propriétés souhaitées des algorithmes de préservation de loi sont obtenues sous ce paradigme. Une part importante de ce chapitre se consacre à décrire et à justifier le modèle de décentralisation que nous avons choisi pour nous permettre de décrire des algorithmes de mise à jour *décentralisés*. Ce chapitre se termine en illustrant la résolution du problème de maintenance, dans le modèle en connaissant la taille du graphe au moment de la mise à jour que nous avons précédemment décrit, sur un exemple simple : les graphes Erdős–Rényi. Une partie de ce chapitre (dont la plupart des définitions) est contenu dans l'article [36], coécrit avec Philippe Duchon.

Le chapitre suivant explore le problème de maintenance sur un modèle *classique* de graphes aléatoires : le *pairing model*, francisé ici en modèle par paire. Ce modèle est utilisé dans la littérature pour obtenir des résultats sur plusieurs classes de graphes aléatoires modélisant des réseaux logiques (réseaux P2P, sociaux, biologiques, etc). Nous proposons de maintenir exactement la distribution du modèle par paire (appariements uniformes), puis de maintenir des (multi)-graphes aléatoires basés sur ce modèle. Nous verrons dans ce chapitre la difficulté de n'utiliser qu'uniquement de l'information locale lors de la maintenance de graphes et l'importance de retenir plus d'information qu'uniquement la topologie du réseau (e.g. boucles, arêtes multiples, orientation des liens). Par exemple, nous fournirons un algorithme semi-décentralisé pour un modèle par attachement préférentiel, basé sur le modèle par paire. Le chapitre se conclut en évoquant la difficulté de maintenir les graphes k-réguliers uniformes, un modèle important de graphes aléatoires.

Dans le chapitre 3, nous explorons en profondeur la maintenance d'une distribution particulière de graphes aléatoires : les graphes k-sortants uniformes. Nous présentons dans un premier temps une argumentation sur la pertinence des graphes k-sortants uniformes en tant que modélisation de réseaux logiques dans un contexte décentralisé. Puis, plusieurs algorithmes sont décrits, dans notre modèle de décentralisation, permettant de maintenir exactement la distribution ciblée. Tous les algorithmes présentés modifient un nombre minimal d'arêtes au cours de la maintenance et accomplissent leur tache en temps moyen constant. Les plus complexes de ces algorithmes permettent de maintenir la loi souhaitée en utilisant un nombre moyen asymptotiquement optimal d'appels à la primitive globale `RandomVertex()`. La fin du chapitre présente des simulations montrant les changements topologiques observés après l'exécution des différents algorithmes présentés ; plusieurs extensions du modèle des graphes k-sortants y sont enfin discutées. Les algorithmes décrits dans ce chapitre et leurs analyses sont publiées dans le travail [36].

Le chapitre 4 aborde la simulation de lois binomiales dans notre modèle décentralisé, un problème soulevé par le chapitre précédent. En effet, dans le chapitre 3, il est supposé un accès direct à la taille du graphe lors de la maintenance et les algorithmes de mise à jour présentés dépendent nécessairement de cette information pour insérer un sommet. Cette dépendance est due à la nécessité de la simulation d'une certaine loi binomiale dépendant du nombre de sommets présents lors de la mise à jour du graphe. Nous présentons dans ce chapitre plusieurs algorithmes de simulation des lois binomiales visées, et ce suivant le modèle d'aléa autorisé. La contribution essentielle de ce chapitre est une nouvelle construction combinatoire sur la génération de permutations. Plus précisément, nous décrivons un nouvel arbre de génération pour les permutations, préservant au maximum le nombre de points fixes. Nous présentons plusieurs applications de cette construction, possédant son intérêt propre en dehors de la maintenance de graphes aléatoires. Une version antérieure et légèrement différente de l'arbre de génération et la description de ces applications est publiée dans l'article [37], coécrit avec Philippe Duchon.

Le dernier chapitre étudie la maintenance de distributions de graphes lorsque la taille n'est pas accessible lors de l'opération de mise à jour. Nous y montrons que le modèle d'Erdős–Rényi n'est pas maintenable, et que les graphes k-sortants uniformes peuvent être maintenus mais en utilisant une quantité plus importante d'aléa global. La différence de maintenabilité est due à la possibilité de simuler une distribution binomiale particulière (abordée au chapitre 4) alors que celle utilisée dans la maintenance du modèle d'Erdős–Rényi n'est pas simulable dans notre contexte. Nous explorons davantage cette notion de simulabilité dans ce chapitre. En outre, nous caractérisons précisément quelles distributions dépendant de la taille du graphe peuvent être générées sans connaître celle-ci, mais en supposant à la place uniquement l'accès à une primitive de type `RandomVertex()`. La fin du chapitre fait le lien entre simulabilité de lois et maintenabilité de distributions de graphes.

Symboles et notations

Nous introduisons dans cette section quelques notations et définitions de base, notamment car pour certaines d'entre elles plusieurs conventions sont d'usage.

Les symboles utilisés pour désigner les différents objets seront souvent réutilisés par la suite, afin de faciliter la lecture.

Ensembles

Nous aurons souvent à ajouter et à enlever un élément d'un ensemble, pour faciliter la lecture nous prenons la liberté de noter l'ajout et la suppression d'un singleton de manière additive : la notation $E+u$ désigne $E \cup \{u\}$ et $E-u$ correspond à $E \setminus \{u\}$.

Lorsque le contexte est clair, l'appartenance sera factorisée, ainsi $a,b,c \in A$ se lit $a \in A$ et $b \in A$ et $c \in A$. La non-appartenance suit la même notation $u,v \notin A$ correspond à $u \notin A$ et $v \notin A$.

La différence symétrique de deux ensembles est notée $A \Delta B = (A \setminus B) \cup (B \setminus A)$.

L'ensemble $\{i, \ldots, j\}$ des entiers naturels compris entre i et j est noté $[\![i, j]\!]$. L'ensemble des entiers non nuls inférieurs à n est noté quant à lui $[\![n]\!] = [\![1, n]\!]$.

La notation $|E|$ désigne le cardinal de E, c'est-à-dire son nombre d'éléments. Lorsque nous écrirons $A \subset E$, l'ensemble A est un sous-ensemble strict ou est égal à E. Un k-sous-ensemble A de E est un sous-ensemble de E de taille k, *i.e.* $A \subset E$ et $|A| = k$.

Graphes

Un graphe G simple (sans boucles ni arêtes multiples) est donné par un couple (V,E) où V est l'ensemble de sommets de G et E l'ensemble de ses arêtes (paires de sommets distincts). Une arête $\{u,v\} \in E$ sera notée uv lorsque la notation n'induit pas de confusion.

Un graphe sur V est un graphe qui possède V comme ensemble de sommets. La taille d'un graphe G est $|V|$, et sa taille totale $|V|+|E|$. Nous utiliserons les notation $V(G)$ pour désigner l'ensemble des sommets d'un graphe G, et $E(G)$ pour désigner l'ensemble de ces arêtes.

Le voisinage (ouvert) $N_G(u)$ d'un sommet u dans G est l'ensemble des sommets *incidents* à v dans G, *i.e.* $N_G(u) = \{v \in V \mid uv \in E\}$. Le voisinage fermé $N_G[u]$ est

obtenu en ajoutant u : $N_G[u] = N_G(u) + u$. Le graphe en indice peut être omis s'il est clair d'après le contexte.

Le degré d'un sommet v noté $\delta(v)$ est la taille de son voisinage ouvert.

Pour $V' \subset V$, on écrira $E|_{V'} = \{uv \in E \mid u,v \in V'\}$ pour l'ensemble des arêtes restreintes au sous-ensemble V'. On note le sous-graphe de G *induit* par V', $G|_{V'} = (V', E|_{V'})$.

Nous notons $d_G(u,v)$ la distance entre les sommets u et v au sein du graphe G, c'est-à-dire la longueur d'un plus court chemin (en nombre d'arêtes) entre u et v dans G ; $d_G(u,v) = \infty$ s'il n'existe pas de chemin reliant u à v. Le diamètre d'un graphe est la plus grande distance apparaissant entre deux sommets, *i.e.* $\mathrm{diam}(G) = \max\{d_G(u,v) \mid u,v \in V\}$. Lorsque le diamètre d'un graphe est fini, le graphe en question est dit *connexe*.

Un graphe G est dit *biparti*, si son ensemble de sommet peut être partitionné en deux sous-ensembles V_1 et V_2, tels que toutes les arêtes de G lient un sommet de V_1 et un sommet de V_2.

Graphes dirigés

La notion de graphe est naturellement étendue lorsque les arêtes possèdent une orientation : un graphe dirigé [5] $G = (V,E)$ a pour ensemble de sommet V et E comme ensemble d'arcs (ou arêtes dirigées), constitué de couples de sommets.

Nous retrouverons un vocabulaire similaire aux graphes non orientés. Ainsi, l'ensemble $N_G^+(u) = \{v \in V \mid (u,v) \in E\}$ est le voisinage sortant du sommet $u \in V$ dans G, et $N_G^+[u] = N_G^+(u) + u$ est le voisinage sortant fermé. Le voisinage entrant de u est noté $N_G^-(u) = \{v \in V \mid (v,u) \in E\}$ et possède aussi son équivalent fermé $N_G^-[u] = N_G^-(u) + u$.

Le degré sortant de v est noté $\delta^+(v) = |N_G^+(v)|$ tandis que son degré entrant est noté $\delta^-(v) = |N_G^-(v)|$.

On parlera de *successeurs* pour les sommets appartenant au voisinage sortant d'un sommet, et de *prédécesseurs* pour ceux appartenant au voisinage entrant.

Probabilités (sur domaines discrets)

La quasi-totalité des distributions de probabilité manipulées dans ce travail seront à support dénombrable. Le support d'une telle distribution de probabilité ρ est noté $\mathrm{Supp}(\rho)$, et correspond aux valeurs possibles pour une variable aléatoire distribuée selon ρ.

La notation $X \sim \rho$ signifie que la variable aléatoire X suit la loi ρ, c'est-à-dire $\mathbb{P}(X = x) = \rho(x)$ pour tout $x \in \mathrm{Supp}(\rho)$.

5. En français, il est parfois d'usage de parler de graphe orienté pour l'anglais *directed graph*. Ici, nous éviterons d'utiliser cette notion car il y a une confusion possible avec l'anglais *oriented graphs* correspondant aux graphes dirigés simples sans boucles et sans circuits de taille 2, ce qui est l'*équivalent* d'orienter un graphe simple.

Nous utiliserons à plusieurs reprises dans des preuves d'indépendance, l'astuce suivante : partant d'un vecteur $(X_i)_{i\in I}$ de variables aléatoires indépendantes, nous décrivons un vecteur $(Y_j)_{j\in J}$ de nouvelles variables aléatoires en tant que fonctions f_j *déterministes* sur des sous-ensembles *déterministes* et *disjoints deux à deux* des X_i, *i.e.* $Y_j = f_j((X_i)_{i\in I_j})$ et $I_j \cap I_{j'} = \emptyset$ lorsque $j \neq j'$. On déduit alors que les $(Y_j)_{j\in J}$ sont indépendants.

Nous parlerons d'évènements E_n arrivant avec forte probabilité lorsque la probabilité que E_n se produise converge vers 1 lorsque n tend vers l'infini. L'expression avec forte probabilité pourra être abrégée par a.f.p.

Le sigle *i.i.d.* est une abréviation d'indépendants et identiquement distribués.

Distributions usuelles

Nous serons amenés à utiliser souvent des distributions de probabilité discrètes usuelles. Nous rappelons ici succinctement leur caractéristiques et les noms que nous utiliserons pour les désigner par la suite.

- **Dir(x)** ou δ_x, la distribution de Dirac en x, est une distribution définie sur un ensemble E où l'élément x, qui appartient à E, a probabilité 1, *i.e.* si p est une Dirac en x sur E, alors $p(x) = 1$ et $p(y) = 0$ pour $y \in E - x$.
- **Ber(p)** : la loi de Bernoulli de paramètre p est une distribution sur {0,1} où 1 a probabilité p et 0 probabilité $1 - p$. Une expérience de Bernoulli de probabilité p est un évènement qui a probabilité p de *succès* et $1 - p$ d'*échec*.
- **Bin(n, p)** : la loi binomiale de paramètres n et p est une distribution sur $[\![0, n]\!]$, telle que l'entier k a probabilité $\binom{n}{p} p^k (1-p)^{n-k}$. Une variable aléatoire qui suit cette distribution peut être vue comme une somme de n Bernoulli Ber(p) *indépendantes*, c'est-à-dire la variable compte le nombre de succès lors de n expériences de Bernoulli indépendantes.
- **Géo(p)** : la loi géométrique de paramètre p est une distribution sur $\mathbb{N}^*$ où k a pour probabilité $(1-p)^{k-1}p$. Une variable aléatoire suivant cette distribution compte le nombre d'essais avant de voir un succès (celui-ci compris) lorsque l'on réalise une série d'expériences de Bernoulli indépendantes.
- **Géo'(p)** est une variante de la distribution Géo(p), définie sur $\mathbb{N}$ où k a pour probabilité $(1-p)^k p$. Cette distribution modélise le nombre d'échecs (potentiellement 0) avant de voir le premier succès lors d'une série d'expériences de Bernoulli indépendantes.
- **Unif(S)** : la loi uniforme sur S est une distribution sur l'ensemble fini S, telle que chaque élément de S est équiprobable, *i.e.* si p est une uniforme sur S, alors $p(s) = 1/|S|$ pour tout $s \in S$.
- **Poi(λ)** : la loi de Poisson de paramètre λ est une distribution sur les entiers où chaque élément k a probabilité $\lambda^k e^{-\lambda}/k!$. Cette distribution modélise le nombre d'évènements arrivant sur une période donnée de temps (ou d'espace), lorsque la moyenne du nombre d'occurrences par unité est λ, et que l'apparition d'un évènement est indépendant de l'intervalle de temps depuis

Nom	Ens. de définition	Espérance	Variance
Ber(p)	{0,1}	p	$p(1-p)$
Bin(n, p)	$[\![0, n]\!]$	np	$np(1-p)$
Géo(p)	$\mathbb{N}^*$	$1/p$	$(1-p)/p^2$
Géo'(p)	$\mathbb{N}$	$(1-p)/p$	$(1-p)/p^2$
Unif($[\![a, b]\!]$)	$[\![a, b]\!]$	$(a+b)/2$	$\frac{(b-a+1)^2-1}{12}$
Poi(λ)	$\mathbb{N}$	λ	λ

TABLE 1 – Résumé des caractéristiques des distributions usuelles.

la dernière apparition.

La table 1 résume les ensembles de définition, moyennes et variances des différentes distributions qui seront utilisées dans ce travail.

Langages

Un mot sur un alphabet Σ est une séquence de lettres de Σ.

Pour un mot w donné, w_i désigne la i-ème lettre du mot w et $|w|$ dénote la longueur du mot w.

Un sous-mot d'un mot w est un mot w' correspondant à une sélection de certaines lettres de w prises dans l'ordre naturel ; pour $n = |w|$, on le définit à partir d'un sous-ensemble I de $[\![n]\!]$ de sorte que $w = w_{i_1} \dots w_{i_{|I|}}$, où $i_1 \dots i_{|I|}$ correspondent aux éléments de I pris dans l'ordre croissant.

Un préfixe d'un mot w est un mot w' tel qu'il existe w'' de sorte que $w = w'.w''$. Un préfixe propre de w est un préfixe w' qui est distinct du mot w lui même, *i.e.* $w' \neq w$.

Algorithmes probabilistes

Si $\mathcal{A}$ est un algorithme, alors la notation $\mathcal{A}(x_1, x_2, \dots, x_\ell)$ désigne la valeur de retour de l'algorithme $\mathcal{A}$ lorsque ses entrées valent respectivement $x_1, x_2, \dots, x_\ell$, ou $\perp$ si $\mathcal{A}$ ne termine pas sur $x_1, x_2, \dots, x_\ell$.

On parlera d'algorithmes *déterministes* pour expliciter le fait que $\mathcal{A}$ n'utilise pas d'aléa pendant ses calculs, c'est-à-dire $\mathcal{A}(x)$ est une fonction déterministe de x, pour $x = x_1, x_2, \dots, x_\ell$.

Lorsqu'une ou plusieurs des entrées $X = X_1, X_2, \dots, X_\ell$ d'un algorithme déterministe $\mathcal{A}$ sont des variables aléatoires, ou lorsque l'algorithme $\mathcal{A}$ fait usage de tirages aléatoires, la valeur $\mathcal{A}(X)$ est elle-aussi une variable aléatoire.

Lorsque $\mathcal{A}$ est déterministe, la distribution de $\mathcal{A}(X)$ est directement liée à la distribution de X : si X suit la loi ρ alors

$$\mathbb{P}(\mathcal{A}(X) = y) = \sum_{x \in \mathrm{Supp}(\rho) \mid \mathcal{A}(x)=y} \rho(x).$$

Un algorithme est dit probabiliste ou *randomisé* si au cours de ses calculs, $\mathcal{A}$ utilise une source d'aléa indépendante de son entrée (que son entrée soit ou non aléatoire).

Afin de modéliser l'aléa utilisé par un algorithme probabiliste, nous supposons qu'il a accès à une primitive `flip()` qui retourne un bit uniforme aléatoire et indépendant des précédents appels. Ce modèle correspond aux machines de Turing probabilistes (voir e.g. [5]), dont une des définitions est l'accès à une fonction `write()` qui écrit un bit aléatoire uniforme sur sa bande de calcul. D'autres définitions équivalentes incluent la possibilité d'utiliser une bande auxiliaire remplie initialement de bits uniformes indépendants, ou rendre le choix uniforme pour la fonction de transition à utiliser dans une machine de Turing non-déterministe possédant deux fonctions de transitions.

Tout algorithme probabiliste $\mathcal{A}$ peut être simulé par un algorithme déterministe $\tilde{\mathcal{A}}$ prenant un mot binaire infini en entrée supplémentaire, et remplaçant le i-ème appel à `flip()` par w_i. Pour un mot binaire fini w, $\tilde{\mathcal{A}}(x; w)$ désigne la valeur de retour de $\tilde{\mathcal{A}}$ lorsque l'algorithme s'arrête en lisant au plus $|w|$ bits et on pose $\tilde{\mathcal{A}}(x; w) = \perp$ dans le cas contraire. La distribution de $\mathcal{A}(X)$ est donnée par :

$$\mathbb{P}(\mathcal{A}(X) = y) = \sum_{n=0}^{\infty} \frac{1}{2^n} \sum_{w \in \{0,1\}^n} \mathbb{P}(\tilde{\mathcal{A}}(X; w) = y).$$

Les algorithmes probabilistes étudiés ici se termineront toujours avec probabilité 1. Si au contraire, on oblige les algorithmes considérés à se terminer quelque soit le résultat des appels à `flip()`, alors même des distributions simples telle que la Bernoulli Ber(1/3) ne peuvent pas être simulées, ce qui est beaucoup trop restrictif.

Méthode de rejet

Une astuce simple afin de générer un élément X uniforme sur un ensemble $A \subset B$ à partir d'un générateur sur un ensemble B, est d'appeler le générateur sur B jusqu'à obtenir un élément de A. On vérifie facilement que chaque élément $a \in A$ est équiprobable :

$$\mathbb{P}(X = a) = \sum_{n=0}^{\infty} \left(1 - \frac{|A|}{|B|}\right)^n \frac{1}{|B|} = \frac{1}{|B|} \cdot \frac{|B|}{|A|} = \frac{1}{|A|}.$$

Cette astuce, appelée méthode de rejet, sera utilisée à plusieurs reprises dans ce travail.

Cette méthode se généralise en utilisant des générateurs sur des ensembles B_1, B_2, …, B_k distincts, tous au moins aussi grand que A, tant que chacun d'entre eux est uniforme et les appels sont indépendants.

Factorielles

La factorielle descendante est notée $(n)_k$ et est définie par

$$(n)_k = \prod_{i=0}^{k-1} n - i = n(n-1)\cdots(n-k+1).$$

La double factorielle $n!!$ est le produit des entiers de même parité que n, *i.e.* $n!! = \prod_{i=1}^{n/2} 2i$ pour n pair et $n!! = \prod_{i=1}^{(n+1)/2} 2i - 1$ pour n impair.

Nous pouvons remarquer que :

— pour $k \geq 0$, $(2k)!! = 2^k k!$ car $(2k+2)!! = 2(k+1)(2k)!!$, et

— pour $k \geq 1$, $(2k-1)!! = (2k)!/(2^k k!)$ car $(2k)! = (2k)!! \cdot (2k-1)!!$.

Permutations

Un ordre sur un ensemble E est une séquence contenant tous les éléments de E (aussi appelé *permutation* sur E). Il existe $|E|!$ tels ordres.

Une permutation de taille n est un des $n!$ ordres sur $[\![n]\!]$. L'ensemble des permutations de taille n est noté dans la suite $\mathcal{S}_n$.

Chapitre 1

Maintenance de graphes aléatoires

Sommaire

Ce chapitre introduit formellement le concept de maintenance de graphes aléatoires étudié dans le reste de cette thèse. Cette maintenance sera effectuée d'un point de vue local, en minimisant l'information globale utilisée par les algorithmes de mise à jour. Le modèle de décentralisation et le formalisme de description des algorithmes ultérieurement présentés y sont notamment décrits. Afin d'illustrer le problème de maintenance, des algorithmes de mise à jour sont ensuite présentés pour le modèle simple et fortement étudié dans la littérature des graphes d'Erdős–Rényi. Ce chapitre nous permet également d'introduire sur des exemples simples, les deux modèles de décentralisation retenus (connaissance ou non de la taille du graphe) et le modèle de coût (global et local) utilisé dans la suite de la thèse.

1.1 Algorithmes de préservation de loi

Commençons par introduire ce que nous appelons *maintenance* de graphe. L'évolution d'un réseau logique est modélisée par une séquence d'opérations dites de *mise à jour*, et correspondant pour chacune d'entre-elles soit à l'ajout d'un nouveau nœud au réseau, soit à la suppression d'un nœud déjà présent dans le réseau. Un protocole de mise à jour décrit comment cette opération doit être effectuée et quelles sont les conséquences sur le réseau.

Afin de modéliser cette procédure, le réseau logique est modélisé par un graphe aléatoire, et le protocole de mise à jour par un algorithme de mise à jour ou algorithme de *maintenance*. Cet algorithme décrit comment doit être effectuée l'insertion ou la suppression d'un sommet du graphe. Nous introduisons dans cette partie la notion d'algorithme de mise à jour, et les propriétés qu'il doit posséder pour *maintenir* effectivement une distribution de graphes aléatoires.

L'ensemble des nœuds possibles du graphe représentant le réseau est noté Ω et sera appelé ensemble sous-jacent ou *univers*. Dans notre contexte, cet ensemble représentera le plus souvent l'ensemble des adresses IP valides, mais pourra être $\mathbb{Q}^2$ pour une approximation d'un espace géométrique bidimensionnel ou un ensemble d'identités valides pour un réseau social. Dans la suite, V représentera toujours un sous-ensemble fini de notre ensemble sous-jacent Ω et sera utilisé pour désigner l'ensemble courant des nœuds de notre réseau, qui évoluera au fur et à mesure des mises à jour (arrivée ou départ d'un nœud du réseau). Son caractère fini sera notamment souvent omis.

L'ensemble des graphes autorisés sur V est noté $\mathcal{G}_V$ et représentera les topologies possibles pour notre réseau logique. Suivant l'application voulue, $\mathcal{G}_V$ peut représenter l'ensemble des graphes simples sur V, graphes orientés sans boucles sur V, idem mais avec boucles, multigraphes sur V avec multiplicité bornée, etc.

Pour les distributions étudiées dans ce travail en particulier, on notera que l'ensemble $\mathcal{G}_V$ est toujours fini. Pour certaines valeurs de V, $\mathcal{G}_V$ peut être vide (par exemple si $|V| < k$ pour une certaine constante k). La suppression de $u \in V$ dans $G = (V,E)$ n'a pas de sens si $\mathcal{G}_{V-u} = \emptyset$, idem pour l'insertion de $w \notin V$ si $\mathcal{G}_{V+w} = \emptyset$. Dans ces situations, aucun comportement spécifique n'est exigé d'un algorithme de mise à jour.

Définition 1.1. Un algorithme d'insertion prend en entrée un graphe $G = (V,E)$ et un sommet $u \in \Omega \setminus V$, et produit en sortie un graphe G' sur $V + u$ (si $\mathcal{G}_{V+u} \neq \emptyset$).

Définition 1.2. Un algorithme de suppression prend en entrée un graphe $G = (V,E)$ et un sommet $u \in V$, et produit en sortie un graphe G' sur $V - u$ (si $\mathcal{G}_{V-u} \neq \emptyset$).

Définition 1.3. Un algorithme de mise à jour est la donnée d'un couple $(\mathcal{A}_i, \mathcal{A}_s)$ composé d'un algorithme d'insertion $\mathcal{A}_i$ et d'un algorithme de suppression $\mathcal{A}_s$.

Ainsi un algorithme d'insertion (resp. suppression) transforme un graphe de $\mathcal{G}_V$ en un graphe de $\mathcal{G}_{V+u}$ (resp. $\mathcal{G}_{V-u}$). Suivant la classe des graphes autorisés, effectuer

une telle transformation efficacement peut déjà poser problème. Dans les différents modèles étudiés ici, cette opération ne pose généralement pas de problème.

L'objectif annoncé de la maintenance de graphes aléatoires est de *conserver* une distribution cible de graphes aléatoires lors de toutes opérations de mise à jour. Cette maintenance permet de résoudre plusieurs problèmes inhérents à l'évolution de réseaux logiques (l'ensemble de ces propriétés est garanti par la proposition 1.1) :

— **Pas de phénomène de dérive** : la distribution du réseau ne s'éloigne pas de la distribution cible au cours des mises à jour. Un réseau, même après un grand nombre d'insertions et de suppressions, aura les mêmes propriétés que le réseau construit en insérant, à partir de zéro, uniquement les nœuds restants après les mises à jour.

— **Analyse simplifiée et propriétés conservées** : la distribution du réseau ne dépend que des sommets présents, voire idéalement que de la taille de cet ensemble (si les identités des sommets ne jouent aucun rôle particulier, cf. § 1.1.1). Une seule distribution par ensemble de sommets (ou taille) doit être analysée pour déduire les propriétés du réseau, indépendamment de son historique d'évolution.

— **Robuste à une séquence adversariale de mises à jour** : la distribution du réseau étant indépendante de la séquence précise de mises à jour, cette dernière peut être laissée à un adversaire. Ainsi, quelle que soit la séquence d'insertions et de suppressions choisie par l'adversaire, la distribution du réseau ne pourra être déviée de celle ciblée. Néanmoins, si l'adversaire a accès au graphe courant, alors aucun algorithme efficace de mise à jour n'est envisageable (voir § 1.1.2).

Pour remplir ces objectifs, les algorithmes considérés seront des algorithmes probabilistes possédant leur propre source d'aléa indépendante du graphe fourni en entrée. Chaque mise à jour produira ainsi un nouveau graphe aléatoire G' à partir d'un graphe aléatoire G, en utilisant potentiellement la source d'aléa interne, *i.e.* appels à une fonction `flip()`, qui est *indépendante* de G. Nous introduisons ci-après la définition précise de cette notion de maintenance de graphes aléatoires.

Définition 1.4. Soit $\mu = (\mu_V)_{V \subset \Omega}$ une famille de distributions de probabilité telle que, pour tout V, le support de μ_V soit inclus dans $\mathcal{G}_V$.

— Un algorithme d'insertion probabiliste $\mathcal{A}$ *préserve* μ si, pour tout $V \subset \Omega$ et pour tout $u \in \Omega \setminus V$, si $\mathcal{G}_{V+u} \neq \emptyset$ et $G \sim \mu_V$ alors $\mathcal{A}(G,u) \sim \mu_{V+u}$.

— Un algorithme de suppression probabiliste $\mathcal{A}$ *préserve* μ si, pour tout $V \subset \Omega$ et pour tout $u \in V$, si $\mathcal{G}_{V-u} \neq \emptyset$ et $G \sim \mu_V$ alors $\mathcal{A}(G,u) \sim \mu_{V-u}$.

Autrement dit, $\mathcal{A}_i$ est un algorithme d'insertion préservant μ si pour chaque sous-ensemble fini $V \subset \Omega$, pour chaque sommet u dans $\Omega \setminus V$, et pour chaque graphe g' dans $\mathcal{G}_{V+u}$, nous avons :

$$\sum_{g \in \mathcal{G}_V} \mu_V(g) \cdot \mathbb{P}(\mathcal{A}(g,u) = g') \;=\; \mu_{V'}(g') \tag{1.1}$$

avec $V' = V + u$. De manière similaire, $\mathcal{A}_s$ est un algorithme de suppression préservant μ si pour tout $V \subset \Omega$, $u \in V$, $g' \in \mathcal{G}_{V-u}$, l'équation 1.1 est vérifiée avec $V' = V - u$. Le couple $(\mathcal{A}_i, \mathcal{A}_s)$ forme alors un algorithme de mise à jour *préservant* la famille de distributions μ.

Appliquer l'algorithme au cours d'un processus de mise à jour, décrit ci-après, offre la garantie que la distribution du graphe obtenu après chaque opération ne dépend que des sommets restants (proposition 1.1) et non de la séquence elle-même.

Exemple 1.1. Prenons l'exemple suivant : un réseau de clients et de serveurs est construit en affectant un serveur uniforme parmi les serveurs présents à chaque client, et ce de manière indépendante. On souhaite maintenir la même distribution de réseau en ajoutant ou supprimant des postes (clients ou serveurs), et si possible en minimisant le nombre de modifications de liens.

Dans notre formalisme, l'univers sera scindé en deux ensembles : celui des clients $(c_i)_{i\in\mathbb{N}}$ et celui des serveurs $(s_i)_{i\in\mathbb{N}}$, soit $\Omega = (s_i)_{i\in\mathbb{N}} \cup (c_i)_{i\in\mathbb{N}}$. Les graphes autorisés $\mathcal{G}_V$ sont les graphes simples sans boucles. Parmi ces graphes, seule une fraction forme le support $\mathcal{S}_V$ de la distribution ciblée sur V, les graphes bipartis sur $V = C \cup S$ où les clients $c \in C$ ont degré 1.

Lorsque V contient k serveurs et $n - k$ clients, chaque graphe $g \in \mathcal{S}_V$ peut être codé par un mot de $n - k$ lettres sur un alphabet de taille k, d'où $|\mathcal{S}_V| = k^{n-k}$. De plus, comme chaque graphe a probabilité $1/k^{n-k}$ d'être obtenu, la distribution cible est donc la loi uniforme ν_V sur $\mathcal{S}_V$.

Afin de supprimer un sommet $u \in V$ d'un graphe $G \sim \nu_V$, nous proposons d'appliquer la procédure suivante :

— Si u est un client, supprimer u et son arête incidente.
— Si u est un serveur, *rediriger* les arêtes incidentes à u vers un des serveurs restants et ce de manière uniforme et indépendante.

Il est aisé de se convaincre que le graphe G' suit la loi ν_{V-u} : si u est un client, il y a k graphes produisant le même graphe de $\mathcal{S}_{V-u}$, et vu que ce nombre ne dépend pas de u ni du graphe obtenu sur $V - u$, la suppression d'un sommet d'un graphe uniforme sur $\mathcal{S}_V$ produit un graphe uniforme sur $\mathcal{S}_{V-u}$; si u est un serveur on a

$$\mathbb{P}(G' = g') = \sum_{a=0}^{n-k} \binom{n-k}{a} \frac{1}{k^{n-k}} \frac{1}{(k-1)^a} = \frac{\left(1 + \frac{1}{k-1}\right)^{n-k}}{k^{n-k}} = \frac{1}{(k-1)^{n-1-(k-1)}}$$

où k est le nombre de serveurs dans V et $g' \in \mathcal{S}_{V-u}$.

De manière similaire, pour ajouter un sommet $u \notin V$ dans un graphe $G \sim \nu_V$, nous pouvons appliquer la procédure suivante :

— Si u est un client, choisir un serveur de V uniformément.
— Si u est un serveur, chaque client choisit le nouveau serveur avec probabilité $\frac{1}{k+1}$.

Pour $g' \in \mathcal{S}_{V+u}$, nous avons bien $\mathbb{P}(G' = g') = 1/k^{n+1-k}$ si u est un client. Sinon, en

notant a le nombre de clients connectés à u, on trouve

$$\mathbb{P}(G' = g') = k^a \frac{1}{k^{n-k}} \frac{1}{k^a} \left(1 - \frac{1}{k+1}\right)^{n-k-a} = \frac{1}{(k+1)^{n+1-(k+1)}}.$$

Ces deux procédures décrivent un algorithme de mise à jour pour ν_V. De plus, le nombre de changement de liens pendant les mises à jour est minimisé : seules des arêtes incidentes au sommet à supprimer ou à insérer ont été supprimées ou ajoutées.

■

Dans la suite, une séquence de mises à jour s sur Ω sera vue comme un mot fini ou infini (séquence infinie de lettre) sur $\Sigma = \{S, I\} \times \Omega$, *i.e.* $s \in \Sigma^*$ ou $s \in \Sigma^\omega$, et où les symboles S et I indiquent le caractère de la mise à jour. Une mise à jour de la forme (S,u) correspond à la suppression du sommet u, et une opération de la forme (I,u) correspond à l'insertion du sommet u.

Par exemple, si à l'état initial, le réseau possède trois nœuds a, b et c, la séquence (S,a), (I,d), (S,c), (I,a) correspond aux mises à jour suivantes : suppression de a, insertion de d, suppression de c, insertion de a. La séquence $((I,d)(S,d))^\omega$ correspond à l'ajout et la suppression en boucle de d.

Notre définition nous garantit qu'étant donné un algorithme $\mathcal{A}$ de mise à jour préservant μ, pour n'importe quel ensemble de départ $V_0 \subset \Omega$, et n'importe quelle séquence de mises à jour s, si notre graphe de départ est distribué selon μ, alors tous les graphes obtenus en suivant la séquence de mises à jour seront correctement distribués.

Plus précisément, ce processus de mise à jour est construit de la sorte : V_0 et s sont fixés au départ, et on suppose G_0 suit la loi μ_{V_0}.

Pour $1 \leq n \leq |s|$, l'ensemble des sommets V_n, constituant le réseau après la n-ième mise à jour, est défini par

$$V_n = \begin{cases} V_{n-1} - u & \text{si } \ s_n = (S,u) \\ V_{n-1} + u & \text{si } \ s_n = (I,u) \end{cases}$$

et le graphe aléatoire G_n, obtenu après la n-ième mise à jour du réseau, est défini par

$$G_n = \begin{cases} \mathcal{A}_s(G_{n-1}, u) & \text{si } \ s_n = (S,u) \\ \mathcal{A}_i(G_{n-1}, u) & \text{si } \ s_n = (I,u). \end{cases}$$

Une séquence de mises à jour s est cohérente avec V_0 si pour tout n, si $s_n = (S,u)$ alors $u \in V_{n-1}$ et $\mathcal{G}_{V_{n-1}-u} \neq \emptyset$; si $s_n = (I,u)$ alors $u \in \Omega \setminus V_{n-1}$ et $\mathcal{G}_{V_{n-1}+u} \neq \emptyset$. Si une séquence n'est pas cohérente avec un certain ensemble V, nous pouvons toujours obtenir une sous-séquence cohérente en ignorant les opérations incohérentes.

Proposition 1.1. *Si $\mathcal{A}$ préserve μ, s est une séquence de mises à jour sur Ω cohérente avec V_0, et G_0 suit la loi μ_{V_0}, alors pour tout n, G_n suit la loi μ_{V_n}.*

Preuve La preuve est une simple induction sur n : le cas de base $G_0 \sim \mu_{V_0}$ est donné en hypothèse, et si $G_n \sim \mu_{V_n}$ alors $G_{n+1} \sim \mu_{V_{n+1}}$ quel que soit s_n car $\mathcal{A}$ est un algorithme de mise à jour préservant μ.

□

La définition choisie, correspond donc bien à notre objectif initial, la distribution du graphe est contrôlée *exactement* à tout moment de l'évolution du réseau, et ce indépendamment de toute stratégie adversariale décidée *a priori*.

1.1.1 Invariance par permutation des sommets

La plupart des familles de distributions que nous allons rencontrer à partir de maintenant vont posséder une propriété dite d'*invariance* par renommage des sommets, *i.e.* la probabilité d'obtenir un graphe en particulier est indépendante des étiquettes des sommets du graphe.

Définition 1.5. Étant donné un graphe $G = (V,E)$ et une bijection $\sigma : V \to V'$, on note le graphe $G[\sigma] = (V',E')$ tel que $uv \in E \Leftrightarrow \sigma(u)\sigma(v) \in E'$.

Une famille de distributions $(\mu_V)_{V\subset\Omega}$, telle que $\mathrm{Supp}(\mu_V) \subset \mathcal{G}_V$, est dite invariante par renommage de sommets si pour tout $V' \subset \Omega$ avec $|V'| = |V|$, et pour toute bijection $\sigma : V \to V'$, si $G \sim \mu_V$ alors $G[\sigma] \sim \mu_{V'}$.

Ce type de distributions se retrouvent typiquement lorsque les nœuds sont traités de manière égalitaire dans le réseau. Au contraire, les familles de distributions basées sur des propriétés géométriques ne sont pas facilement capturées par invariance : le plus souvent l'identité d'un nœud correspond à sa position dans l'espace, et un graphe aléatoire *géométrique* est construit à partir des distances entre ces nœuds. La maintenance exacte (et efficace) de ces modèles pose un défi particulier dans un contexte dynamique et décentralisé, qui ne sera pas abordé au cours de cette thèse.

Exemple 1.2. La distribution de graphes aléatoires de l'exemple 1.1 n'est clairement pas invariante par renommage de sommets : un serveur (dont le degré est potentiellement supérieur à 1) ne joue pas le même rôle qu'un client et *vice versa*. En revanche, elle est invariante par les renommages qui permutent les serveurs entre eux et les clients entre eux.

Un exemple simple de distribution présentant cette propriété est la distribution des circuits aléatoires uniformes, où un circuit est un cycle orienté. Cette distribution peut facilement être maintenue de la manière suivante :

- Insertion de u : choisir un sommet uniforme v du circuit, et ajouter u entre v et son successeur w, *i.e.* supprimer l'arc (v,w) et ajouter les arcs (v,u) et (u,w).
- Suppression de u : rediriger le prédécesseur v de u vers son successeur w, *i.e.* supprimer les arcs (v,u) et (u,w) et ajouter l'arc (v,w).

Il existe $(n-1)!$ tels circuits avec n sommets. L'insertion d'un sommet produit bien n nouveaux circuits (choix de v), et $(n-1)$ circuits avec n sommets produisent le même circuit avec un sommet de moins (choix du prédécesseur de u).

De manière un peu plus générale, les permutations, vues comme des graphes orientés avec boucles, peuvent de même être maintenues de manière similaire : il suffit d'ajouter à l'insertion la possibilité d'insérer u en point fixe avec probabilité $1/(n+1)$. Leur capacité à modéliser des réseaux est toutefois plus contestable. ■

Pour les distributions invariantes par renommage, le raisonnement peut se faire typiquement en terme de nombre de sommets et l'analyse peut se concentrer sur $\mathcal{G}_n$, les graphes sur $[\![n]\!]$. Nous continuerons néanmoins à maintenir une distribution μ_V sur $\mathcal{G}_V$ afin que :

— la suppression d'un sommet d'identité u garde le sens qu'on lui ait donné et n'a pas de conséquences sur le reste des identités, l'objectif étant d'obtenir un graphe sur $V-u$ et non supprimer n dans $\mathcal{G}_n$ ou supprimer $u \in [\![1,n]\!]$ et *renommer* les autres sommets ;

— lorsque aucune suppression n'est effectuée, les identités des nœuds de $\mathcal{G}_n$ n'indiquent pas clairement leur ordre d'insertion.

1.1.2 Modèle d'adversaires forts

Il est important de noter que dans la définition donnée à la section précédente, bien que le graphe d'entrée soit aléatoire, le sommet à supprimer ou insérer ne l'est pas, *i.e.* la distribution de $\mathcal{A}(G,u)$ est garantie pour tout choix *déterministe* de u.

Ceci est important en ce qui nous concerne, car nous souhaitons laisser potentiellement le choix de la prochaine mise à jour à un adversaire. En particulier, comme u est déterministe, il ne peut dépendre du graphe G qui lui est aléatoire : par exemple un adversaire ne peut choisir de supprimer un sommet de plus petit ou de plus fort degré dans G.

En fait, u peut bien-entendu être choisi aléatoirement tant que sa distribution est *indépendante* du graphe courant. En particulier, un adversaire peut faire la requête d'appliquer un ensemble de mises à jour donné $U \subset \Omega$ sans préciser l'ordre dans lequel les opérations doivent être effectuées. Dans ce cas de figure, un algorithme de maintenance peut choisir un ordre arbitraire pour les opérations, puis effectuer les mises à jour dans cet ordre ; en respectant l'équation 1.1, le graphe obtenu à la fin de la séquence est garanti de suivre la distribution appropriée. Notamment, si un ordre sur V est déjà connu, aucun aléa supplémentaire n'est nécessaire, sinon un ordre uniforme pourra être choisi. Un exemple de cette situation est donné dans la section 2.2.3.

Proposition 1.2. *Soit $\mathcal{A}$ un algorithme d'insertion (resp. suppression) préservant μ, $V \subset \Omega$ et $U \subset \Omega \setminus V$ (resp. $U \subset V$) et G un graphe distribué selon la loi μ_V. Si $\mathcal{G}_{V \cup W} \neq \emptyset$ (resp. $\mathcal{G}_{V \setminus W} \neq \emptyset$) pour tout $W \subset U$, et S est un ordre aléatoire sur U*

indépendant de G, alors l'itération de l'algorithme $\mathcal{A}$ sur G avec $S_1 \dots S_{|U|}$ engendre un graphe distribué selon $\mu_{V \cup U}$ (resp. $\mu_{V \setminus U}$).

Preuve Soit $\mathcal{A}(g,U;\sigma) = \mathcal{A}(\dots \mathcal{A}(g,\sigma_1),\dots,\sigma_{|U|})$ le graphe final obtenu après les $|U|$ insertions successives de sommets selon l'ordre σ, en partant du graphe $g \in \mathcal{G}_V$.

Comme G et S sont indépendants, la probabilité d'obtenir le graphe $g' \in \mathcal{G}_{V \cup U}$ est

$$\sum_{\sigma \text{ ordre sur } U} \mathbb{P}(S = \sigma) \sum_{g \in \mathcal{G}_V} \mathbb{P}(G = g) \cdot \mathbb{P}(\mathcal{A}(g,U;\sigma) = g')$$

Or, nous pouvons conclure car comme $\mathcal{A}$ préserve μ, la deuxième somme est $\mu_{V \cup U}(g')$ pour tout U et ordre σ sur U (proposition 1.1).

La suppression est obtenue de manière analogue.

□

Une question se pose alors : le problème de maintenance est-il toujours intéressant lorsque l'adversaire peut avoir accès au graphe ? Ce modèle plus fort d'adversaire ne l'est malheureusement pas car les seuls algorithmes de mise à jour possibles ne peuvent pas être considérées comme *efficaces* dans notre modèle, c'est à dire sous-linéaires.

Un algorithme de mise à jour est appelé *trivial* s'il ne tient pas compte de l'état actuel du graphe et construit un nouveau graphe en sortie indépendamment de celui passé en entrée. Autrement dit, $\mathcal{A} = (\mathcal{A}_i, \mathcal{A}_s)$ est *trivial* si pour tout $V \subset \Omega$ et pour tout $g \in \mathcal{G}_V$, nous avons

— $\forall u \in \Omega \setminus V$, $\forall g' \in \mathcal{G}_{V+u}$, $\mathbb{P}(\mathcal{A}_i(g,u) = g') = \mu_{V+u}(g')$, et

— $\forall u \in V$, $\forall g' \in \mathcal{G}_{V-u}$, $\mathbb{P}(\mathcal{A}_s(g,u) = g') = \mu_{V-u}(g')$.

Un tel algorithme correspond exactement au problème de génération aléatoire (voir § 1.2.5), lorsqu'on ne tient pas compte de la décentralisation du système. Le coût de chaque mise à jour (insertion et suppression) sera alors au minimum proportionnel à la taille totale du graphe d'arrivée.

Nous allons prouver que seul un algorithme trivial peut préserver μ lorsque l'adversaire choisit la prochaine mise à jour selon l'état courant du réseau. Plus précisément, après une unique modification du réseau, la distribution du graphe de sortie s'est déjà potentiellement éloignée de celle ciblée. On supposera pour simplifier que l'adversaire choisit *déterministiquement* la prochaine mise à jour (l'opération et le sommet impliqué) pour chaque graphe d'entrée possible. Un tel adversaire est *sans-mémoire*, et est nécessairement plus faible qu'un adversaire qui pourrait avoir accès à l'ensemble des graphes obtenus lors du processus de mise à jour.

Soit V un ensemble fixe de sommets et $G \sim \mu_V$ un graphe aléatoire représentant le réseau à un temps donné. Un adversaire (pour l'ensemble V) est modélisé par une fonction $\mathrm{Adv} : \mathcal{G}_V \to (\{S\} \times V) \cup (\{I\} \times \Omega \setminus V)$ tel que $\mathrm{Adv}(g)$ indique la prochaine mise à jour à effectuer lorsque le graphe courant est g. Vu que le graphe initial est aléatoire, l'opérande de la mise à jour l'est aussi, et donc notre définition actuelle d'algorithme de mise à jour ne s'applique pas : u n'est clairement pas une variable aléatoire dans la définition 1.4.

Une définition cohérente est d'exiger de la sortie de l'algorithme d'être distribuée selon μ_{V+u} conditionnellement à $\mathrm{Adv}(G) = (I,u)$ ou selon μ_{V-u} conditionnellement à $\mathrm{Adv}(G) = (S,u)$.

Définition 1.6. Un algorithme de mise à jour $\mathcal{A} = (\mathcal{A}_i,\mathcal{A}_s)$ préserve μ dans un *modèle d'adversaires forts* si pour tout $V \subset \Omega$, et pour tout adversaire Adv, si $G \sim \mu_V$ et $\mathrm{Adv}(G) = (O,U)$ alors :

— $\mathcal{A}_i(G,U)|_{O=I,U=u} \sim \mu_{V+u}$ et $\mathcal{A}_s(G,U)|_{O=S,U=u} \sim \mu_{V-u}$.

Sous cette définition, seuls des algorithmes triviaux préservent μ.

Proposition 1.3. *Un algorithme de mise à jour $\mathcal{A}$ préservant μ dans un modèle d'adversaires forts est nécessairement trivial.*

Preuve Supposons que $\mathcal{A}$ n'est pas trivial, *i.e.* il existe un ensemble de sommets V où $\mathcal{A}$ tient compte de son entrée, plus précisément il existe $g_1 \in \mathcal{G}_V$ et il existe (O,u) de sorte que :

— soit $(O,u) = (S,u)$ tel que $u \in V$ et $\mathbb{P}(\mathcal{A}(g_1,u) = g_1') \neq \mu_{V-u}(g_1')$ pour un certain $g_1' \in \mathcal{G}_{V-u}$,

— soit $(O,u) = (I,u)$ tel que $u \in \Omega \setminus V$ et $\mathbb{P}(\mathcal{A}(g_1,u) = g_1') \neq \mu_{V+u}(g_1')$ pour un certain $g_1' \in \mathcal{G}_{V+u}$.

On construit alors l'adversaire suivant : $\mathrm{Adv}(g_1) = (O,u)$ et $\mathrm{Adv}(g) = (I,v)$ avec $v \in \Omega \setminus V$ pour tout $g \in \mathcal{G}_V - g_1$.

$\mathcal{A}$ ne préserve pas μ pour l'adversaire Adv.

□

De plus, pour tout algorithme non trivial, il existe une stratégie adversariale pour obtenir un graphe qui ne suit pas la distribution visée. Cette stratégie se base sur le plus petit ensemble V où l'algorithme $\mathcal{A}$ tient compte de son entrée. Précisément, partant d'un graphe de départ sur V_0, l'adversaire supprime tous les sommets de V_0, puis insère tous les sommets de V afin d'obtenir un graphe G sur V (moins d'opérations sont nécessaires lorsque $V \cap V_0 \neq \emptyset$). Si $G = g$, l'adversaire applique l'opération (O,u) de la proposition précédente, et dans le cas contraire supprime n'importe quel sommet v de V et obtient le graphe G'. Comme $\mathcal{A}$ est trivial sur $V - v$, l'insertion de v dans G' produit un graphe distribué selon μ_V quelque soit G'. L'adversaire répète alors la suppression puis l'insertion de v jusqu'à obtenir g ; ce processus termine avec probabilité 1 comme $g \in \mathrm{Supp}(\mu_V)$.

Lors d'un processus de mise à jour sur une séquence infinie de mises à jour choisies par l'adversaire décrit précédemment, la distribution de G_n conditionné à $V_n = V + u$ (ou $V - u$ si $O = S$) est infiniment souvent différente de μ_V.

1.1.3 Une notion de dualité entre algorithmes

Nous achevons cette section en présentant une notion de dualité pour les algorithmes de mise à jour. Deux algorithmes, un d'insertion et un de suppression, sont

dits *duaux* lorsqu'ils définissent la même loi jointe du couple (G,G') où $G \in \mathcal{G}_V$ et $G' \in \mathcal{G}_{V+u}$ pour tout $V \subset \Omega$ et $u \notin V$. Pour un algorithme d'insertion, ce couple de variables aléatoires correspond à l'entrée G et à la sortie G' de l'algorithme (après insertion de u) ; pour un algorithme de suppression, il correspond à la sortie G et à son entrée G' (suppression du sommet u).

Notamment, deux algorithmes sont duaux pour une certaine famille de distribution μ si l'un est l'*inverse* de l'autre pour μ : c'est-à-dire si la probabilité de passer d'un graphe g à g' se déduit entièrement de l'autre algorithme et de μ. Cette notion est à rapprocher de la notion de réversibilité des chaînes de Markov.

Définition 1.7. Un algorithme d'insertion $\mathcal{I}$ et un algorithme de suppression $\mathcal{D}$ sont *duaux* pour $\mu = (\mu_V)_{V\subset\Omega}$ avec $\mathrm{Supp}(\mu_V) \subset \mathcal{G}_V$, si pour tout $V \subset \Omega$, $u \in \Omega \setminus V$, $g \in \mathcal{G}_V$, $g' \in \mathcal{G}_{V+u}$, nous avons :

$$\mu_V(g) \cdot \mathbb{P}(\mathcal{I}(g,u) = g') = \mu_{V+u}(g') \cdot \mathbb{P}(\mathcal{D}(g',u) = g).$$

Des algorithmes duaux ont le même comportement vis-à-vis de la préservation de loi.

Proposition 1.4. *Si $\mathcal{I}$ et $\mathcal{D}$, respectivement un algorithme d'insertion et un de suppression, sont duaux pour μ, alors soit ils préservent tous deux μ, soit aucun des deux ne préserve μ.*

Preuve Soient $V \subset \Omega$ et $u \in \Omega \setminus V$.

Supposons $\mathcal{I}$ préserve μ, on a alors pour $g \in \mathcal{G}_V$:

$$\sum_{g'\in\mathcal{G}_{V+u}} \mu_{V+u}(g') \cdot \mathbb{P}(\mathcal{D}(g',u) = g) = \mu_V(g) \sum_{g'\in\mathcal{G}_{V+u}} \mathbb{P}(\mathcal{I}(g,u) = g') = \mu_V(g),$$

donc $\mathcal{D}$ préserve bien μ.

Si $\mathcal{I}$ ne préserve pas μ, il existe $g \in \mathcal{G}_V$ tel que la somme précédente est différente de $\mu_V(g)$, et donc $\mathcal{D}$ ne préserve pas μ.

Le cas symétrique se prouve de manière analogue.

□

Enfin, notons que la définition implique directement l'égalité des supports des lois jointes définies par les algorithmes de mise à jour.

Proposition 1.5. *Soit $(\mathcal{I},\mathcal{D})$ un algorithme de mise à jour pour $\mu = (\mu_V)_{V\subset\Omega}$. Si $\mathcal{I}$ et $\mathcal{D}$ sont duaux, alors pour tout $V \subset \Omega$, $u \in V \setminus \Omega$, $g \in \mathcal{G}_V$ et $g' \in \mathcal{G}_{V+u}$:*

$$\mathbb{P}(\mathcal{I}(g,u) = g') > 0 \Leftrightarrow \mathbb{P}(\mathcal{D}(g',u) = g) > 0.$$

Cette notion de dualité peut être utile de deux manières différentes :

— Afin de prouver plus aisément la correction (voire la non-correction) d'algorithmes de maintenance. Elle sera notamment utilisé dans cette perspective dans le § 1.4.2 et aussi exploitée au § 3.18.

— Afin de concevoir des algorithmes de préservation de lois. Une fois une des deux opérations de mise à jour trouvée, nous obtenons une distribution duale dont nous savons qu'elle possède la propriété de préservation de loi. La dernière étape consiste alors à trouver un algorithme produisant cette loi conditionnelle entre son entrée et sa sortie.

1.2 Modèle de décentralisation

L'objectif de ce travail est de maintenir des distributions de graphes aléatoires, tout particulièrement dans un contexte décentralisé. La décentralisation du système sera modélisée de la manière suivante :

— les nœuds peuvent communiquer avec leur voisinage,
— les nœuds ont accès à une primitive `RandomVertex()`, renvoyant un nœud uniforme (décrite ci-dessous).

Notre modèle de maintenance se divise en deux sous-modèles :

— tous les nœuds connaissent la taille *exacte* du réseau.
— la taille du réseau est inconnue des nœuds, et une borne inférieure ne peut être acquise qu'en répétant les appels à `RandomVertex()` ou en parcourant le réseau.

Suivant les distributions de graphes à maintenir, l'information globale que constitue la taille du réseau soit sera inutile (modèles par paires, chapitre 2), soit permettra d'économiser des appels à la primitive globale `RandomVertex()` (graphes k-sortants uniformes, chapitre 3), soit sera même nécessaire afin de maintenir la distribution considérée (graphes d'Erdős–Rényi § 1.4 et chapitre 5).

La taille d'un réseau de type P2P est par nature une information globale qu'*a priori* les nœuds ne sont pas censés connaître et *a fortiori* une valeur exacte de la taille est vite périmée au vu du fort dynamisme des réseaux P2P. Néanmoins, différentes techniques permettent d'obtenir une estimation fiable de la taille, voir e.g. l'étude [69] qui compare expérimentalement plusieurs d'entre elles. L'estimation peut aussi être calculée de manière décentralisée et sécurisée afin que tous les nœuds possèdent une estimation [45]. Dans d'autres situations où le réseau est moins volatile ou possède un contrôle central (plateformes de calcul collaboratives, base de donnée distribuée), l'accès à la taille est plus réaliste.

1.2.1 La primitive RandomVertex

L'appel à la primitive `RandomVertex()` (que l'on abrégera souvent par `RV`) fournit l'identité d'un sommet du réseau choisi aléatoirement et uniformément dans l'ensemble des nœuds présents dans le réseau, exception faite du nœud à l'origine de la mise à jour.

Définition 1.8. L'utilisation de `RandomVertex()` par un algorithme de mise à jour d'un graphe $G = (V,E)$ renvoie :

— un sommet uniforme de $V - u$, lors de la suppression de u ;

— un sommet uniforme de V, lors de l'ajout de u dans G ;

et les appels successifs à la fonction renvoient des résultats indépendants.

Cette définition correspond au cas général de graphes étudiés ici, c'est-à-dire des graphes sans boucles. Elle permet dans ce cas, de sélectionner un sommet pouvant *remplacer* u lors de la suppression, et un sommet pouvant appartenir au voisinage de u lors de l'insertion, et cela sans faire de rejets. Lorsque les graphes maintenus peuvent contenir des boucles, il peut être plus intéressant de supposer que `RV` travaille directement sur $V + u$ à la place de V lors de l'insertion de u. D'un point de vue de la simulabilité, générer un sommet uniforme sur $V + u$ peut tout à fait être simulé via cette procédure mais cela entraîne un surcoût lorsque la taille du graphe est inconnue (voir `RandomVertex(V+u)`, § 1.3.2).

Dans tout réseau décentralisé évoluant dans le temps, il existe un mécanisme (intrinsèquement *externe* au réseau) afin de contacter un des nœuds déjà présents dans le réseau. Le protocole décrivant l'ajout d'un nouveau nœud démarre ainsi *stricto sensu* après une opération externe au protocole qui consiste à *contacter un nœud ami*, et souvent aucune hypothèse particulière n'est émise par rapport à ce premier nœud contacté (voir l'étude [1]). Dans les implémentations de réseaux P2P, il est d'usage qu'un serveur (parmi plusieurs miroirs) contenant les adresses d'un certain nombre de *pairs* potentiels soit contacté pour amorcer l'insertion dans le réseau.

Par exemple, les protocoles Gnutella [53] et Freenet [26] ne spécifient pas la procédure pour rejoindre le réseau décentralisé. Les procédures typiquement utilisées par les différents clients font intervenir un serveur afin de récupérer l'adresse d'un nœud (cf. [1]). Un exemple typique de ce genre de comportement est le protocole BitTorrent [27] où une phase centralisée est préalable (requête sur un des serveurs d'un fichier recherché) à la phase décentralisée (téléchargement du fichier), voir [83]. Le réseau GNUnet [55] spécifie entièrement comment un premier nœud peut être contacté lors de la première connexion au réseau (à partir de serveurs webs appelés *HOSTLIST* hébergés par les nœuds du réseau le souhaitant).

L'hypothèse que le sommet renvoyé par ce mécanisme externe est uniforme sur le réseau est une hypothèse forte qui ne se retrouve pas tel quel dans la littérature. Sa justification est due à plusieurs problèmes posés par la maintenance exacte de graphes aléatoires :

— La primitive `RV` doit être **randomisée** : si `RV` est déterministe, l'aléa *global* ne peut être obtenu qu'en acquérant une certaine connaissance du réseau (expliqué § 1.2.5). Si la fonction est déterministe, en plus du goulot d'étranglement que serait alors le nœud contacté, il n'est pas possible de maintenir (efficacement) la plupart des familles de distributions de graphes intéressantes.

— La primitive `RV` doit être **globale**, *i.e.* tous les nœuds doivent être accessibles. Si certains nœuds ne sont jamais renvoyés par la primitive, leur identité ne

pourra être acquise qu'en parcourant le réseau, potentiellement en temps $\Omega(|V|)$, et leur présence même ne pourra jamais être confirmée si on ne connaît pas la taille du graphe.

Il faut bien voir dans cette primitive globale une idéalisation des différents processus de *premier contact* qui sont utilisés dans les réseaux de types P2P aujourd'hui. Plusieurs mécanismes similaires apparaissent dans la littérature, sélectionnant des nœuds aléatoirement dont la distribution est proche de l'uniforme, sous certaines conditions ; nous en décrivons certains au prochain paragraphe.

Sachant que la majorité des distributions étudiées ici reste invariante par renommage des sommets, la distribution de `RV` n'a pas de motivation pour dépendre des identités des nœuds. Deux distributions simples peuvent être envisagées respectant ces critères : uniforme sur l'ensemble des nœuds, ou proportionnelle au degré (de façon générale, dépendant du voisinage à distance k). Nous discutons de ces deux distributions dans la suite de cette section.

Implémentations possibles

Dans la littérature, nous pouvons retrouver des mécanismes similaires à notre primitive `RV`. De manière non exhaustive, nous décrivons certains de ces mécanismes qui supposent tous un premier contact arbitraire.

Dans le protocole *pair à pair* Chord [89], les nœuds forment une topologie en anneau et implémentent une table de hachage distribuée. Le système de requête du réseau permet de retrouver le nœud responsable d'une clé (e.g. correspondant à un fichier) en temps $\mathcal{O}(\log(n))$, avec forte probabilité, où n est la taille du réseau. Suivant le principe du hachage cohérent [61], chaque nœud se voit attribuer une proportion approximativement identique de l'espace des clés et peu de réallocations de clés sont à effectuer lors de l'évolution du réseau. Ainsi, trouver le nœud responsable d'un *hache* généré uniformément permet de sélectionner un nœud aléatoire du réseau dont la distribution s'approche de l'uniforme.

Un deuxième moyen de sélectionner un tel nœud est de passer directement par les *finger table* en décidant avec équiprobabilité la portion de l'anneau où sera situé le nœud à sélectionner, puis en remontant d'un niveau dans la table. Par exemple, au premier tour, le nœud u décide s'il doit contacter un nœud avant ou après v, le nœud stocké dans le dernier niveau de sa *finger table*. La distribution du nœud obtenu par de tels processus de sélection est néanmoins dépendante du réseau.

Dans le réseau décentralisé à faible diamètre de [76], les auteurs supposent explicitement l'existence d'un serveur contenant l'identité de K nœuds du réseau, où K est une constante prédéfinie. Lorsqu'un nouveau nœud souhaite intégrer le réseau, il contacte le serveur afin d'obtenir un ensemble de D nœuds uniformes parmi les K nœuds présents dans le cache du serveur. Le nombre de connexions simultanées que chaque nœud du réseau peut acquérir par la suite, est aussi borné par une autre constante. Au cours de l'évolution du réseau, les différents nœuds passent régulièrement dans le cache du serveur (sous certaines hypothèses sur la dynamicité du

réseau). Ainsi le cache du serveur représente un sous-ensemble aléatoire de taille K du réseau, où tous les nœuds passent régulièrement. Les appels successifs au serveur ne sont toutefois pas indépendants (les nœuds restent dans le cache jusqu'à obtenir un nombre prédéfini de connexions).

Le protocole SWAN [23] définit un processus d'évolution de réseau sous la forme d'un graphe k-régulier (avec k pair) construit aléatoirement. Lorsqu'un nœud rejoint le réseau, $k/2$ marches aléatoires sont démarrées afin de sélectionner $k/2$ sommets dans le graphe, et en supposant les marches assez longues pour que les sommets sélectionnés soient *presque* uniformes. Chaque sommet u sélectionné choisit alors une arête uv uniforme dans son voisinage, qui sera remplacée par deux arêtes reliant u et v au nouveau sommet. Les connexions perdues inopinément se récupèrent en broadcastant sur le réseau pour chercher un autre sommet de degré inférieur à k. Dans l'analyse [28] de l'évolution du réseau vu comme une chaîne de Markov, il est montré que son temps de mélange est borné polynomialement en N, l'espérance du nombre de sommets, sous certaines hypothèses sur la dynamicité du réseau.

Un autre exemple où des marches aléatoires sont utilisées se trouve dans le protocole présenté dans [29]. Le protocole repose sur des jetons, contenant chacun l'adresse d'un nœud du réseau. Les jetons accomplissent une marche aléatoire sur le graphe jusqu'à être utilisés. Lorsqu'un nouveau nœud u souhaite rejoindre le réseau, il récupère des jetons à partir d'un nœud ami jusqu'à en obtenir k différents, où k est une constante prédéfinie, et correspondant effectivement à des nœuds présents dans le réseau (les jetons pouvant être périmés). Le processus d'insertion se termine lorsque u donne à son tour à ses voisins un nombre constant de jetons portant son identité. Les jetons sont aussi utilisé pour *réparer* les connexions perdues. Après un temps raisonnable, de l'ordre de $\mathcal{O}(\log(n))$, les jetons sont distribués de sorte que chacun constitue un sommet *presque* uniforme. Sous certaines hypothèses concernant l'évolution du réseau, la connectivité peut être maintenue face à une suppression adversariale des liens de communication.

Enfin, dans un autre contexte de réseaux logiques que sont les réseaux de calculs collaboratifs, il peut être envisagé de supposer qu'il existe un nœud répartiteur qui a connaissance des adresses des différents nœuds de calcul. Dans ce cas, une sélection uniforme ne pose pas de problème.

1.2.2 Pouvoir de simulation de RandomVertex

Octroyer l'accès à une primitive globale tirant un sommet aléatoire du graphe, et ce uniformément et de manière indépendante, fournit un pouvoir de simulation important aux algorithmes de mise à jour, exécutés localement sur les nœuds du graphe. Nous présentons ici quelques méthodes de simulation et problématiques inhérentes à la génération aléatoire dans un contexte décentralisé. Les différentes méthodes présentées ici seront notamment utilisées au cours de cette thèse.

Apprentissage de V

Lorsque la taille du graphe est connue du point de vue des algorithmes de mise à jour, alors l'ensemble des sommets bien qu'inconnu peut être acquis. En effet, vu que le nombre exact de sommets est connu, on peut appeler la primitive globale jusqu'à acquérir l'identité de tous les sommets du graphes. Il est connu qu'il faut en moyenne $\mathcal{O}(n \log(n))$ tirages d'uniformes sur V avant d'obtenir le set complet V (problème du collectionneur de coupons). Une fois l'ensemble V acquis, une génération classique peut démarrer (voir § 1.2.5).

Néanmoins, une telle simulation entraînera un coup prohibitif de mise à jour pour la structure. La complexité d'acquérir V est au moins linéaire dans notre modèle (e.g. en parcourant le graphe).

Tirage uniforme d'arête

La possibilité de tirer uniformément un sommet nous permet aussi d'obtenir une arête uniforme, lorsque l'on a la garantie que le graphe possède au moins une arête.

Dans le cas général, une méthode simple consiste à tirer deux sommets via `RV` jusqu'à obtenir une paire reliée par une arête. Chaque arête est obtenue de façon équiprobable (méthode par rejet), et il faut en moyenne n^2/m tirages de `RV` pour obtenir une arête uniforme (pour un graphe à n sommets et m arêtes). Donc, cette méthode n'a besoin que d'un nombre de tirage constant asymptotiquement pour les graphes denses, possédant $\Omega(n^2)$ arêtes.

Dans le cas plus favorable, où le degré des sommets est borné par une constante k, $k/\tilde{\delta}$ tirages en moyenne sont suffisant pour tirer une arête uniforme ; ici, $\tilde{\delta} = 2m/n$ est le degré moyen. La méthode consiste alors à tirer uniformément un sommet u puis sélectionner une arête uniforme dans son voisinage avec probabilité $\delta(u)/k$ et avec probabilité complémentaire, on relance la méthode.

Enfin, si on ne possède pas la certitude que le graphe possède bien des liens (un cas peu probable mais qui peut arriver sur certaines distributions de graphes aléatoires), alors il n'est possible de tirer une arête uniforme (ou retourner *échec* s'il n'y en a aucune) qu'en ayant la connaissance de la taille du graphe. Même avec cette information, l'algorithme devra passer en revue le voisinages de tous les sommets pour se convaincre qu'il n'existe aucune arête dans le graphe.

Tirage proportionnel au degré des sommets

Une alternative à `RandomVertex()`, est de supposer la possibilité de tirer un sommet du graphe proportionnellement à son degré à la place d'uniformément. Une telle primitive peut être particulièrement appropriée dans des modèles de graphes par attachement préférentiel (cf section 2.3).

Pour cela, il suffit de tirer une arête, puis choisir une de ses deux extrémités avec probabilité 1/2 chacune. La probabilité d'obtenir un sommet v donné est alors $\delta(v)/(2m)$.

Dans le cas où on connaît une borne sur les degrés des sommets, la méthode de tirage d'arêtes uniformes revient à tirer un sommet uniforme v et de le *conserver* avec probabilité $\delta(v)/(2k)$, avec probabilité $\delta(v)/(2k)$ sélectionner à la place une des extrémités de ces arêtes incidentes uniformément, et avec probabilité complémentaire $1 - \delta(v)/k$ rejeter.

Économie de tirages

Une méthode qui peut sembler efficace pour *économiser* le recours à la primitive globale est d'utiliser les sommets déjà connus et la connaissance de la taille du graphe. Si un ensemble W de k sommets du graphe sont connus à un certain moment de l'algorithme, afin de tirer un sommet uniforme sur V, on peut choisir chacun des sommets déjà connus avec probabilité $1/n$, et avec probabilité complémentaire tirer un sommet de $V \setminus W$.

Donc, avec probabilité k/n, on ne fera aucun appel à la primitive globale. Avec probabilité $1-k/n$, il faut choisir un sommet *inconnu* par l'algorithme de l'ensemble V. Comme le seul moyen d'acquérir l'identité d'un nouveau sommet est d'appeler la primitive `RV`, il faudra en moyenne $n/(n-k)$ appels afin d'obtenir un sommet en dehors de l'ensemble W (voir la fonction `RandomVertexAvoiding` § 1.3.2).

Sauf dans le cas où on a déjà acquis tous les sommets ($n = k$), au total, 1 appel à `RV` en moyenne sera utilisé, ce qui rend complètement inutile un tel mécanisme.

RandomVertex(V+u)

Nous avons choisi de supposer que la primitive globale travaille sur l'ensemble des sommets V lors de l'insertion d'un nouveau sommet $u \notin V$. Ce choix est essentiellement motivé, d'une part car les distributions étudiées dans ce travail n'ont pas d'intérêt à pouvoir choisir u via `RV`, et d'autre part car octroyer la possibilité à la primitive globale de renvoyer comme *nœud ami*, l'initiateur de l'appel, perd un peu de sens dans notre contexte d'évolution de réseaux logiques.

Nous montrons ici que ce choix n'entrave pas le pouvoir de simulation des algorithmes de mise à jour : un sommet uniforme sur $V + u$ peut toujours être simulé dans notre modèle, et ce en moins de deux appels à `RV` en moyenne.

Pour la simulation de `RandomVertex(V+u)`, deux modèles de décentralisation peuvent être envisagées, suivant la connaissance de n, la taille de V.

Si on a accès à n, il suffit de renvoyer u avec probabilité $1/(n+1)$, et avec probabilité complémentaire, d'appeler `RV` afin d'obtenir un sommet uniforme sur V. Moins d'un appel à `RV` est nécessaire pour tout n, et la complexité de l'algorithme est constante.

Si la taille de V est inconnue, il n'est pas possible de simuler `RandomVertex(V+u)` en au plus un appel à la primitive `RV`.

Nous proposons l'algorithme suivant afin de simuler `RandomVertex(V+u)` :

$v_0 \leftarrow$ **RandomVertex**()
répéter
  $b \leftarrow$ *Bernoulli*(1/2) ; $v \leftarrow$ **RandomVertex**()
jusqu'à $b = 0$ ou $v = v_0$
si $b = 0$ **alors**
  retourner v
sinon
  retourner u

Conditionné à v_0, l'algorithme est un algorithme par rejet sur $V \times \{0,1\}$ et accepte uniquement l'ensemble $V \times \{0\} + (v_0,1)$. Donc chaque élément x de l'ensemble $V + u$, a probabilité $1/(n+1)$ d'être retourné. En moyenne $2n/(n+1) \in [1,2]$ tirages sont nécessaires, et la complexité moyenne est constante.

Enfin, nous aurions très bien pu prendre la convention inverse, *i.e.* `RV` travaille directement sur $V + u$ lors de l'insertion. Dans ce cas, un sommet uniforme sur V est simulé en asymptotiquement un tirage en tirant des sommets sur $V + u$, jusqu'à en obtenir un différent de u (voir `RandomVertexAvoiding`, § 1.3.2).

1.2.3 Aspects décentralisés

Dans un but de clarté, l'ensemble des algorithmes étudiés ici seront présentés de manière séquentielle et centralisée. Une telle description va nous permettre de prouver que le graphe aléatoire obtenu en sortie de l'algorithme respecte bien la distribution souhaitée, et ce de manière plus aisée qu'en fournissant l'implémentation précise de l'algorithme dans un modèle par passage de messages.

Néanmoins, pour rester dans l'objectif initial fixé, nous prendrons bien soin à ne pas utiliser l'ensemble des sommets du graphe courant : les algorithmes de mise à jour ne pourront communiquer qu'avec des nœuds dont ils ont préalablement appris l'identité. Précisément, on se restreint à deux opérations pour apprendre l'identité d'un nœud du réseau :

— soit en faisant directement appel à la primitive `RandomVertex()`,
— soit en demandant à un nœud déjà connu de fournir une partie de son voisinage.

On suppose ainsi qu'un nœud pourra être contacté à partir du moment où on acquiert son identité, ce qui correspond tout à fait aux réseaux P2P, où typiquement l'identité est l'adresse IP du nœud en question.

À partir du moment où nos algorithmes respectent ces principes, il n'est pas difficile de voir comment les implémenter dans un modèle décentralisé synchrone et sans pannes, par passages de messages de taille non-bornée (appelé dans la littérature modèle $\mathcal{LOCAL}$ [77]), augmenté de la primitive `RV`. Nous ne considérerons pas ici les problèmes pouvant être liés aux pannes, à des opérations concurrentes ou à des nœuds malveillants, qui sont en dehors du champ d'étude de ce travail.

La taille des messages ne sera par défaut pas bornée, mais dans les algorithmes présentés ici, leur taille sera typiquement de l'ordre de $\log_2(|V|)$, afin de coder les identités des nœuds impliqués. Les liens de communications seront toujours considérés comme symétriques. En particulier, l'orientation des arcs dans un graphe dirigé n'est que purement symbolique au niveau de la communication : un nœud peut communiquer avec ses voisins entrants et sortants. Les boucles et arêtes multiples n'ont pas d'effet sur les possibilités de communication.

Exemple 1.3. Les procédures décrites afin de maintenir la distribution des circuits aléatoires uniformes (exemple 1.2) sont par nature locales : uniquement de l'information locale est utilisée (excepté un sommet uniforme pour l'insertion), et les modifications de topologies sont aussi purement locales.

Par exemple, nous pouvons convertir la description centralisée en une procédure par passage de messages de la manière suivante :

— **Insertion de** u : u utilise la primitive globale `RV` pour avoir accès à l'identité d'un sommet v, puis envoie une requête à v afin d'obtenir l'identité du successeur actuel w de v, puis u contacte v et w afin d'établir un lien logique orienté de v vers u et u vers w, et de supprimer le lien logique existant v vers w ; enfin u met à jour son état interne pour représenter les nouveaux liens logiques. En termes de messages, cela se traduit par la procédure suivante :

$v \leftarrow$ `RandomVertex()`
$w \leftarrow u$.send(v, GET)
u.send(v, $\rightarrow$, u)
u.send(w, $\leftarrow$, u)
u.send(v, $\not\rightarrow$, w)
u.send(w, $\not\leftarrow$, v)

en supposant la commande *send*(destinaire, message, paramètre) pour envoyer les messages, et les messages GET (obtention du voisin sortant), $\rightarrow$ / $\not\rightarrow$ ajout/suppression de lien sortant, et de manière symétrique $\leftarrow$ / $\not\leftarrow$ ajout/suppression de lien entrant. Les messages sont interprétés en mettant à jour l'état interne des nœuds ou renvoyant un message contenant l'identité d'un voisin.

— **Suppression de** u : u contacte son prédécesseur v et lui demande de supprimer le lien logique le liant à u et de le remplacer par un lien vers w le successeur de u. En gardant les messages utilisés précédemment, on obtient la procédure suivante :

$v \leftarrow u$.prédécesseur
$w \leftarrow u$.successeur
u.send(v, $\not\rightarrow$, u)
u.send(w, $\not\leftarrow$, u)
u.send(v, $\rightarrow$, w)
u.send(w, $\leftarrow$, v)

■

Dans la suite, les transcriptions dans un modèle par passage de messages, comme présentées dans l'exemple précédent, ne seront pas fournies.

1.2.4 Modèle de coût

L'efficacité des algorithmes sera calculée sur deux critères distincts : le coût de décentralisation et la complexité temps.

Le **coût de décentralisation** (ou *coût global*) est la charge d'information globale utilisée par l'algorithme et correspond au nombre d'appels à la primitive `RV` effectués au cours d'une mise à jour. Sachant que `RV` concentre l'information globale utilisée par l'algorithme, et que chaque appel peut être vu comme une procédure lourde cassant momentanément la décentralisation du système, le nombre d'appels à la primitive doit être minimisé. Dans la suite, le *coût* d'un algorithme de mise à jour désigne son coût de décentralisation. En pratique, nous nous intéresserons particulièrement au coût moyen asymptotique d'une mise à jour.

La **complexité temps** (ou *coût local*) correspond au nombre de pas de calculs nécessaires à une opération de mise à jour pour s'effectuer. Dans la description centralisée des algorithmes, elle correspond à la notion traditionnelle de *complexité* pour l'algorithme considéré, en considérant qu'un appel à `RV` possède un coût unitaire. Pour maintenir efficacement une distribution de graphes, la complexité temps doit être idéalement faible, c'est-à-dire en $o(|V|)$. Idéalement, une mise à jour s'effectuera en temps $\mathcal{O}(1)$, ou au pire $\mathcal{O}(\log(|V|))$. Dans la suite, la *complexité* d'un algorithme de mise à jour désigne sa complexité temps. Ici, nous nous intéresserons uniquement à la complexité moyenne d'une mise à jour. Précisément, nous nous plaçons dans un modèle de coût unitaire pour les opérations arithmétiques de base (manipulant des sommets) et certains générateurs aléatoires. Ce modèle correspond aux machines RAM w-bits en faisant l'hypothèse que $\log_2(|V|) \leq w$, ce qui est largement raisonnable au vu de la taille des réseaux actuels (au maximum de l'ordre de la dizaine de millions de nœuds) et la taille du mot mémoire (32 ou 64 bits), cf. e.g. [93, 81]. Dorénavant, les complexités exprimées seront toujours à comprendre dans ce modèle RAM w-bits, où $w \geq \log_2(n)$ pour n la taille du graphe courant.

Un algorithme de mise à jour est considéré comme efficace si sa complexité moyenne et son coût moyen asymptotique sont tous les deux faibles. On cherchera notamment à atteindre un coût moyen asymptotique optimal, tout en conservant une complexité constante, lorsque cela est possible.

1.2.5 Liens avec la génération aléatoire exacte

La génération aléatoire exacte consiste à trouver des algorithmes probabilistes afin de construire des objets d'une certaine taille et selon une loi fixée à l'avance. Suivant les objets à générer et les lois considérées, cette simulation peut se faire de manière plus ou moins efficace.

Nous considérons ici la génération aléatoire exacte de graphes aléatoires selon μ_V, voire μ_n si la distribution ne dépend que de la taille du graphe.

Un algorithme de mise à jour pour une certaine famille de distributions $(\mu_V)_{V\subset\Omega}$ de graphes aléatoires fournit directement une méthode de génération aléatoire des objets pour μ_V, à partir du moment où

— une méthode est connue pour les petites tailles (par exemple un ensemble de sommets $V_0 \subset V$), et

— il est possible de former $V_1 \subset V_2 \cdots \subset V_\ell = V$ en ajoutant successivement des sommets à V_0 de telle que sorte que l'ensemble des graphes autorisés $\mathcal{G}_{V_i}$ n'est pas vide, et ce pour tout $i \in [\![\ell]\!]$.

Par exemple, si on sait maintenir $(\mu_n)_{n\geq 1}$ où μ_n est une distribution sur les graphes simples $\mathcal{G}_n$ à n sommets, alors pour construire un graphe selon μ_n, il suffit de partir de $G_1 \sim \mu_1$ et d'insérer $2, \ldots, n$ successivement à G_1. Par la propriété de préservation de distribution (proposition 1.1), on a $G_n \sim \mu_n$. En particulier, si la procédure de maintenance consomme un temps moyen constant, la procédure de génération pour G_n est en moyenne en $\mathcal{O}(n)$.

La réciproque n'est pas nécessairement vraie. Si une méthode est connue pour générer des objets selon μ_n, rien ne contraint cette méthode : elle ne crée pas forcément des objets en augmentant leur taille, les objets intermédiaires ne sont pas forcément bien distribués, la méthode peut être difficile à décentraliser, et la méthode ne fournit pas forcément un moyen simple de supprimer un sommet.

Exemple 1.4. Notons K_V le graphe complet sur l'ensemble V, et considérons le graphe aléatoire G_V sur V suivant

$$G_V = \begin{cases} K_V & \text{si } |V| \text{ est pair} \\ K_{V-U} & \text{si } |V| \text{ est impair} \end{cases}$$

où U est un sommet uniforme de V (qui restera alors isolé).

Il est très facile de générer un graphe selon la distribution décrite : lorsque $n = |V|$ est pair, ajouter les $\binom{n}{2}$ arêtes possibles pour construire K_V, lorsque n est impair tirer un sommet u uniforme sur V et ajouter les $\binom{n-1}{2}$ arêtes possibles entre sommets de $V - u$ pour construire K_{V-u}.

Lorsque l'on connaît n, ou au moins sa parité, il n'est pas difficile de maintenir cette distribution. Par exemple, pour insérer un nouveau sommet, il faudra localiser le sommet isolé lorsque $|V|$ est impair (temps linéaire en moyenne) pour pouvoir le connecter au reste du graphe.

Si la taille de l'ensemble V n'est pas connue, n'importe quel algorithme de mise à jour est forcément incorrect. À l'insertion de u dans V, soit l'algorithme n'augmente jamais le nombre d'arêtes (incorrect pour $|V|$ impair), soit l'algorithme l'augmente de temps en temps (incorrect pour $|V|$ pair).

■

Si la taille du graphe est connue, alors une méthode de génération pour μ_n pour tout n fournit directement un algorithme trivial pour préserver μ : quelle que soit son entrée, la méthode de génération est utilisée avec comme nombre de sommets $n \pm 1$ selon l'opération. Autrement dit, une reconstruction depuis zéro est effectuée, et comme déjà mentionné dans le paragraphe 1.1.2, ces algorithmes ne sont pas satisfaisants. De même, si la distribution de graphes aléatoires considérée ne contient que des graphes connexes, le nombre de sommets peut être acquis en parcourant le graphe.

Si la taille du graphe est inconnue, une méthode pour maintenir $(\mu_n)_{n\geq 1}$ peut être impossible alors qu'il existe bien une méthode pour construire G_n selon la loi μ_n. Des exemples de telles familles de distributions sont donnés dans le chapitre 5.

1.3 Aléa interne et externe

Afin de décrire dans les chapitres suivants des algorithmes de mise à jour pour la tâche de maintenance, nous définissons précisément ici quelques générateurs aléatoires, qui seront utilisés dans la description des algorithmes.

Ces générateurs seront de deux catégories distinctes :

— les générateurs dits **internes** qui ne feront jamais appel à `RV`. Ces générateurs peuvent tous être implémentés en supposant l'accès à une source de bits aléatoires uniformes et indépendants.

— les générateurs dits **externes** qui utiliseront tous `RV`. Ces générateurs nous serviront de *macro* afin de simplifier l'écriture des algorithmes de mise à jour.

On pourra noter que si la garantie nous est fournie que le réseau possède au moins deux nœuds, alors même les générateurs **internes** peuvent être implémentés via des appels à `RV` (cette simulation est décrite précisément § 5.2.1). Ce cas de figure ne sera néanmoins pas pris en compte car l'objectif annoncé des algorithmes de mise à jour est de minimiser la quantité d'aléa *globale* utilisée, qui serait mis en échec directement si `RV` devait aussi être utilisé afin de simuler la source d'aléa interne.

1.3.1 Générateurs internes

Nous allons être amené à utiliser les générateurs aléatoires suivants, qui reprennent les distributions déjà présentées dans le chapitre de symboles et notations :

— *Bernoulli*(p) renvoie 1 avec probabilité p et 0 avec probabilité $1-p$. Le résultat de ce générateur sera utilisé pour réaliser des tests booléens (où 1 correspond à *vrai*, et 0 à *faux*).

— *Binomiale*(n,p) retourne $k \in [\![0, n]\!]$ avec probabilité $\binom{n}{p} p^k (1-p)^{n-k}$.

— *Uniforme*(S) retourne chaque élément de S avec probabilité $1/|S|$.

— *Permuter*(S) retourne l'ensemble S dans un ordre aléatoire uniforme.

Chaque appel à l'un des générateurs fournit une variable aléatoire indépendante des précédents appels au même générateur, ainsi que des appels aux autres générateurs. En effet, chaque appel peut être vu comme consommant une séquence finie de bits aléatoires uniformes indépendants fournis par une source inépuisable.

Dans la suite, nous aurons à calculer le coût moyen de différents algorithmes utilisant ces générateurs de base. Comme cette thèse ne s'intéresse pas à comment implémenter efficacement ces différents générateurs, nous ferons les hypothèses suivantes :

- Le générateur ***Bernoulli***(p) prend un temps moyen $\mathcal{O}(1)$ lorsque $p \in \mathbb{Q}$. Ce cas est le seul rencontré dans ce travail, et sa complexité vient du fait qu'on peut calculer efficacement le développement binaire de p (voir [51, 25]).
- Le générateur ***Binomiale***(n,p) prend un temps moyen proportionnel à np pour $n = |V|$ et $n = \Omega(\log(1/p))$; en particulier lorsque $np = \mathcal{O}(1)$, le générateur prend un temps moyen de $\mathcal{O}(1)$. On trouve plusieurs algorithmes dans [33] simulant une telle distribution en temps moyen de l'ordre de np dans un modèle *Real RAM* [14], e.g. à partir de géométrique Géo(p), ou de variables exponentielles. D'après [25], une géométrique Géo(p) peut être simulée en temps $\mathcal{O}(1 + \log(1/p)/w)$ sur une machine RAM w-bits avec $w = \Omega(\log\log(1/p))$, ce qui implique la borne fournie. En pratique, des algorithmes plus efficaces peuvent être utilisés (voir [60, 71]) mais, en ce qui nous concerne, la borne donnée est suffisante.
- Le générateur ***Uniforme***(S) prend un temps moyen de $\mathcal{O}(1)$ car un indice dans le *tableau* S peut être généré en temps constant (dans notre modèle de machine) ; de manière générale, il faut $\mathcal{O}(\log(|S|))$ bits pour générer un des indices de S.
- Le générateur ***Permuter***(S) a une complexité de $\mathcal{O}(|S|)$ en moyenne ; ce coût correspond à la complexité de générer une permutation aléatoire uniforme de S.

En particulier, les deux derniers générateurs ont des complexités en $\mathcal{O}(1)$ lorsque $|S| = \mathcal{O}(1)$.

1.3.2 Générateurs externes

Afin de raccourcir l'écriture des algorithmes de mise à jour, nous définissons quelques fonctions, basées sur la primitive d'aléa global `RandomVertex()`. Contrairement aux générateurs de la section précédente, la source principale d'aléa utilisée ici sera `RV`, et le coût considéré sera essentiellement évalué en terme de nombre moyen d'appels à la primitive.

Les complexités moyennes des fonctions seront toujours du même ordre de grandeur que leur nombre d'appels à la primitive globale. Ces fonctions n'auront pas besoin d'accéder à la taille n de V.

RandomVertexAvoiding(W)

La fonction retourne un sommet uniforme de $V \setminus W$ et est implémentée ainsi :

répéter
  $x \leftarrow$ **RandomVertex**()
jusqu'à $x \notin W$
retourner x

La fonction sera toujours appelée avec un ensemble W de sorte que $V \setminus W$ n'est pas vide. L'implémentation correspond à un algorithme par rejet sur V afin de générer un élément du plus petit ensemble $V \setminus W$. Le nombre moyen d'appels à `RV` est distribué géométriquement avec comme moyenne $n/(n-k)$ pour $|W| = k$, ce qui correspond à un unique appel asymptotiquement dès lors que $k = \mathcal{O}(1)$.

De plus, comme les appels successifs à `RV` sont indépendants, les appels successifs à **RandomVertexAvoiding** avec des paramètres indépendants, sont eux-même indépendants.

RandomSubset(k)

La fonction retourne un sous-ensemble de V de taille k, uniformément choisi, et la fonction est implémentée de la sorte :

$S \leftarrow \emptyset$
répéter
  $S \leftarrow S+$ **RandomVertexAvoiding**(S)
jusqu'à $|S| = k$
retourner S

Lors de l'appel à la fonction, on prendra soin d'avoir $|V| \geq k$. Si on note S_i le i-ème élément ajouté à S, la probabilité d'obtenir S peut se calculer ainsi pour $\mathcal{S} \subset V$ avec $|\mathcal{S}| = k$:

$$\mathbb{P}(S = \mathcal{S}) = \sum_{s \text{ ordre sur } \mathcal{S}} \mathbb{P}(S_1 = s_1, S_2 = s_2, \ldots, S_k = s_k).$$

Par définition de **RandomVertexAvoiding**, $\mathbb{P}(S_1 = s_1) = 1/n$. Conditionné à $S_1 = s_1$, S_2 reçoit un élément uniforme sur $V - s_1$, donc $\mathbb{P}(S_2 = s_2 \mid S_1 = s_1) = 1/(n-1)$. En itérant ce raisonnement, on trouve

$$\mathbb{P}(S = \mathcal{S}) = k! \cdot \frac{1}{n} \cdots \frac{1}{n-(k-1)} = \frac{1}{\binom{n}{k}}.$$

En moyenne, le nombre d'appels à la primitive globale `RV` est donné par

$$\sum_{i=0}^{k-1} \frac{n}{n-i} = n \cdot (H_n - H_{n-k}),$$

où H_n est le nombre harmonique d'ordre n, *i.e.* $H_n = \sum_{i=0}^{n} 1/i$.

Notons que cette somme est bornée par $k\frac{n}{n-k}$, soit $k + \mathcal{O}(1/n)$ lorsque k est constant.

RandomSubsetAvoiding(W, k)

La dernière fonction combine les deux précédentes : elle renvoie un sous-ensemble uniforme de taille k de $V \setminus W$, implémentée ainsi :

$S \leftarrow \emptyset$
répéter
  $S \leftarrow S+$ **RandomVertexAvoiding**$(S \cup W)$
jusqu'à $|S| = k$
retourner S

Bien entendu, l'appel à cette fonction ne se fera que lorsque $|V \setminus W| \geq k$. Nous pouvons appliquer le même raisonnement que pour la fonction précédente mais en remarquant que les éléments sélectionnés appartiennent forcément à $V \setminus W$. Son coût est similaire à la fonction précédente : $k + \mathcal{O}(1/n)$.

1.4 Maintenance des graphes d'Erdős–Rényi

Nous illustrons dans cette section la maintenance de graphes aléatoires, dans notre modèle de décentralisation, sur un modèle simple et classique : le modèle $G(n,p)$ ou graphes d'Erdős–Rényi de densité fixe.

Un graphe aléatoire G suit la distribution $G(n,p)$, dite distribution des graphes d'Erdős–Rényi, si G possède n sommets étiquetés et chaque arête possible de G est présente avec probabilité p indépendamment des autres. Ici, p est constant et n'évoluera pas avec n. On note que la probabilité d'obtenir un graphe donné ne dépend alors que de son nombre d'arêtes.

Le modèle a été introduit en premier par Gilbert [54], à la suite d'un article [42] d'Erdős et Rényi sur un modèle légèrement différent. La simplicité du modèle et le fait que les arêtes sont présentes indépendamment les unes des autres permettent d'obtenir des résultats très précis sur plusieurs paramètres intéressants : connectivité [42], degrés des sommets [17], taille des plus importantes composantes connexes (voir e.g. [43, 44, 18]), diamètre, etc (voir [19] entre autres). Les graphes d'Erdős–Rényi sont un des modèles de graphes aléatoires les plus étudiés et permettent de répondre à de nombreuses questions en théorie des graphes (voir e.g. [22]). Par exemple, une de leurs applications s'inscrit dans ce qu'on nomme la *méthode probabiliste*, et permet de prouver l'existence d'un objet combinatoire en travaillant uniquement sur les graphes aléatoires (cf. les papiers précurseurs d'Erdős [40, 41, 39] qui fournissent des bornes inférieures sur les nombres de *Ramsey*).

Afin d'illustrer le problème de maintenance, nous nous proposons de maintenir la (famille de) distribution $G(n,p)$ face à des insertions et suppressions de sommets. Comme dans le problème de maintenance, les identités des nœuds doivent être conservées au cours des différentes insertions et suppressions, nous considérons le graphe aléatoire $G(V,p)$ suivant le modèle d'Erdős–Rényi sur l'ensemble V et nous avons $G(n,p) = (G(V,p))_{V \subset \Omega}$, où Ω est notre univers.

Définition 1.9. Un graphe aléatoire simple G suit la distribution $G(V,p)$ si

$$\mathbb{P}(G = (V,E)) = p^{|E|}(1-p)^{\binom{|V|}{2}-|E|}.$$

Dans la suite, $V \subset \Omega$ et $n = |V|$. Les graphes autorisés $\mathcal{G}_V$ sont tous les graphes simples d'ensemble de sommets V.

1.4.1 Génération aléatoire

Avant de s'intéresser à maintenir une distribution de graphes au cours du temps, il est important de savoir comment générer de tels graphes à taille fixée, comme énoncé dans le paragraphe 1.2.5.

Une manière simple de générer un graphe $G \sim G(V,p)$ est de parcourir l'ensemble des arêtes possibles, et pour chacune d'entre elle, générer une Bernoulli $\mathrm{Ber}(p)$ nous indiquant si l'arête est présente ou absente. La complexité moyenne de ce générateur simple est de $\Theta(n^2)$ car toutes les $\binom{n}{2}$ arêtes possibles doivent être passées en revue.

Une méthode généralement plus efficace [46, 10], en particulier lorsque p n'est pas proche de 1, consiste à générer des variables géométriques de paramètre p qui vont compter combien d'arêtes vont être *sautées* avant que la prochaine ne soit ajoutée au graphe. L'algorithme devra effectuer autant de tirages que d'arêtes présentes dans le graphe final. Si les géométriques sont obtenues chacune en temps constant, alors on dispose d'un générateur en temps moyen $\mathcal{O}(n+m)$, où $m = \Theta(pn^2)$ est le nombre moyen d'arêtes.

Nous savons, au moins depuis Devroye [33], qu'il est possible en temps constant de générer de telles géométriques si on dispose d'un générateur d'uniformes sur $[0,1]$ et d'une machine à arithmétique réelle (modèle *Real RAM*). Ces algorithmes sont facilement transposables sur des machines physiques actuelles au prix de remplacer les réels par des nombres à virgule flottante, ce qui aura pour effet dans notre cas, de biaiser légèrement la distribution du graphe de sortie.

Il est aussi possible d'atteindre la même performance et une génération exacte [25] dans un modèle plus réaliste de machines, les *RAM* w-bits, que nous utilisons pour calculer la complexité temps de nos algorithmes de maintenance, en supposant $w = \Omega(\log(n))$.

1.4.2 Algorithmes de maintenance

Au vu de la simplicité du modèle $G(n,p)$, et notamment de l'indépendance des arêtes entre elles, il est facile de trouver comment mettre à jour ces graphes aléatoires. Néanmoins, nous proposons d'expliciter ces algorithmes dans le modèle particulier de décentralisation qui est le nôtre (*i.e.* où l'ensemble des sommets n'est pas accessible et l'aléa global est restreint à une primitive `RV`) afin d'illustrer le problème de maintenance, et notre méthodologie de preuve. Nous supposerons ici que nous avons connaissance de n, la taille du graphe lors de l'opération de mise à jour.

Algorithme 1 SUPPRESSION-GNP

Entrées: un graphe $G = (V, E)$ et un sommet $u \in V$
Sortie: un graphe G'
1: $V' \leftarrow V - u$
2: $E' \leftarrow E|_{V'}$
3: $G' \leftarrow (V', E')$

Algorithme de suppression

Supprimer un sommet u dans $G \sim G(V,p)$, de telle sorte que le graphe obtenu G' après suppression soit distribué comme $G(V-u,p)$ est une opération aisée : il suffit de supprimer u et toutes les arêtes incidentes à u dans G. Le graphe obtenu suit bien la distribution ciblée car le graphe aléatoire $G|_{V-u}$ est indépendant du voisinage de u. Nous allons prouver ce résultat en comptant les graphes $g \in \mathcal{G}_V$ produisant $g' \in \mathcal{G}_{V-u}$, pondéré par la probabilité que $G = g$, illustrant une méthodologie de preuve qui sera utilisée à plusieurs reprises par la suite.

Proposition 1.6. *L'algorithme* SUPPRESSION-GNP *préserve la distribution* $G(n,p)$.

Preuve Soient $G \sim G(V,p)$ le graphe avant la mise à jour, G' le graphe obtenu après la suppression du sommet u de V, g' un graphe de $\mathcal{G}_{V-u}$ et $n = |V|$.

Comme pour tout algorithme probabiliste, nous pouvons calculer la probabilité d'obtenir g' en sortie par la formule suivante :

$$\mathbb{P}(G' = g') = \sum_{g \in \mathcal{G}_V} \mathbb{P}(G = g) \cdot \mathbb{P}(G' = g' \mid G = g). \tag{1.2}$$

Comme l'algorithme est déterministe, $\mathbb{P}(G' = g' \mid G = g) = 1$ si et seulement si $g' = g|_{V-u}$ est la restriction à $V - u$ de g. Il existe 2^{n-1} graphes de $\mathcal{G}_V$, qui satisfont $g' = g|_{V-u}$, correspondant au nombre de voisinages possible pour u. De plus, comme u a degré k dans $\binom{n-1}{k}$ de ces graphes, nous obtenons :

$$\begin{aligned}
\mathbb{P}(G' = g') &= \sum_{k=0}^{n-1} \sum_{N \subset V, |N|=k} \mathbb{P}(G = (V, E(g') \cup N)) \\
&= \sum_{k=0}^{n-1} \binom{n-1}{k} p^{|E(g')|+k} (1-p)^{\binom{n}{2} - |E(g')| - k} \\
&= p^{|E(g')|} (1-p)^{\binom{n-1}{2} - |E(g')|} \underbrace{\sum_{k=0}^{n-1} \binom{n-1}{k} p^k (1-p)^{n-1-k}}_{1}
\end{aligned}$$

où on reconnaît les probabilités de la loi $\mathrm{Bin}(n-1, p)$ dans la dernière somme.

□

L'algorithme qui a été donné (voir algorithme 1) représente la version minimaliste de ce qui est demandé pour toute procédure de maintenance, et qui apparaîtra dans la description des prochains algorithmes de mise à jour : l'ensemble des sommets est mis à jour (augmentation ou diminution d'un élément) et par défaut l'ensemble des arêtes E' est restreint au nouvel ensemble de sommets, c'est-à-dire :

— $E' = E - N(u)$ lors de la suppression de u,

— $E' = E$ lors d'une insertion.

À partir de cet ensemble par défaut, la procédure spécifie les arêtes à supprimer ou à ajouter à l'ensemble E'. Dans le cas des graphes $G(n,p)$, aucune modification supplémentaire n'est à apporter.

Algorithme d'insertion

Algorithme 2 Insertion-Gnp

Entrées: un graphe $G = (V, E)$ et un sommet $u \notin V$
Sortie: un graphe G'

1: $V' \leftarrow V + u$
2: $E' \leftarrow E$
3: $k \leftarrow \mathit{Binomiale}(n,p)$
4: $N \leftarrow \textbf{RandomSubset}(k)$
5: **pour tout** $v \in N$ **faire**
6: $\quad E' \leftarrow E' + \{u,v\}$
7: $G' \leftarrow (V', E')$

En ce qui concerne l'insertion, même la méthode de génération naïve, qui consiste à passer en revue chaque arête possible, n'est pas effective dans notre modèle car V n'est pas accessible. Comme ici, on suppose qu'on a accès à n, V peut être obtenu par des tirages successifs à `RV` au prix d'une complexité au moins linéaire en n (comme expliqué § 1.2.2).

Pour obtenir un algorithme plus efficace, il suffit de séparer la génération du degré du nouveau sommet à insérer et le choix de son voisinage. En effet, nous savons que le degré de n'importe quel sommet dans un graphe d'Erdős–Rényi suit la loi binomiale avec paramètres $n - 1$ et p. Une fois son degré k choisi, on peut construire le voisinage de u via un appel à la fonction `RandomSubset(`k`)`.

Comme les arêtes (et *non-arêtes*) entre des sommets de V n'ont pas été modifiées, on peut facilement se convaincre que la probabilité d'obtenir un graphe donné $g = (V + u, E)$ est bien $p^{|E|}(1-p)^{\binom{|V+u|}{2}-|E|}$.

Proposition 1.7. *L'algorithme* Insertion-Gnp *préserve la distribution* $G(n,p)$.

Preuve On suppose $G \sim G(V,p)$, $n = |V|$, $u \in \Omega \setminus V$, $g \in \mathcal{G}_V$ et $g' \in \mathcal{G}_{V+u}$. Nous utilisons la même stratégie que la preuve de la proposition 1.6.

Comme seul un graphe $g = g'|_V$ dans $\mathcal{G}_V$ peut produire le graphe g', l'équation 1.2 se simplifie à

$$\mathbb{P}(G' = g') = \mathbb{P}(G = g) \cdot \mathbb{P}(G' = g' \mid G = g).$$

Notons $k = \delta_{g'}(u)$. Par définition du générateur Binomiale(n,p) et de la fonction `RandomSubset(k)` (dont les résultats sont indépendants), on trouve

$$\begin{aligned} \mathbb{P}(G' = g') &= p^{|E(g)|}(1-p)^{\binom{n}{2}-|E(g)|} \cdot \binom{n}{k} p^k (1-p)^{n-k} \cdot \frac{1}{\binom{n}{k}} \\ &= p^{|E(g')|}(1-p)^{\binom{n+1}{2}-|E(g')|} \end{aligned}$$

où $|E(g')| = |E(g)| + k$.

□

Proposition 1.8. *L'algorithme* INSERTION-GNP *utilise en moyenne $\mathcal{O}(n)$ appels à la primitive `RV`, et possède une complexité temps en $\mathcal{O}(n)$ en moyenne.*

Preuve La complexité de l'algorithme, en nombre de pas de calculs, est proportionnelle au nombre d'appels à `RV` effectués lors de l'exécution de `RandomSubset(k)`, ligne 4 de l'algorithme. Ce nombre d'appels $\mathcal{R}$, est quant à lui en moyenne linéaire.

Pour s'en rendre compte, notons que nous pouvons borner son espérance par

$$\mathbb{E}(\mathcal{R}) \;\leq\; \mathbb{P}(B < np') \cdot \mathbb{E}(\mathcal{R} \mid B < np') + \mathbb{P}(B \geq np') \cdot \mathbb{E}(\mathcal{R} \mid B \geq np')$$

où B est le résultat de l'appel à *Binomiale*(n,p) ligne 3 et $p < p' < 1$.

De plus, nous avons

$$\mathbb{P}(B < np') \cdot \mathbb{E}(\mathcal{R} \mid B < np') \leq n(H_n - H_{n-np'})$$

en utilisant l'analyse de `RandomSubset` au § 1.3.2. Cette contribution apporte donc $n \log(1/(1-p')) + \mathcal{O}(1/n)$ à l'espérance de $\mathcal{R}$.

Concernant $\mathbb{P}(B \geq np')$, une simple inégalité de Bienaymé-Tchebychev nous fournit la borne suivante

$$\mathbb{P}(B \geq np') \leq \frac{1}{n} \frac{p(1-p)}{(p'-p)^2}.$$

Enfin remarquons que $\mathbb{E}(\mathcal{R} \mid B \geq np')$ est bornée par l'espérance du collectionneur de coupons avec n coupons soit $n \cdot H_n$.

En ajoutant les deux contributions nous obtenons bien $\mathbb{E}(\mathcal{R}) = \mathcal{O}(n)$.

□

Sans connaissance de la taille

On notera que la taille de V a été utilisée lors de la génération de la variable binomiale. Si la taille du réseau n'est pas accessible aux nœuds, alors il n'existe pas d'algorithmes de maintenance pour le modèle $G(n,p)$ (la preuve est donnée dans la section 5.1.2).

Dualité des algorithmes

Remarquons finalement comment nous aurions aussi pu utiliser la notion de dualité, présentée § 1.1.3, afin de prouver la qualité de conservation de lois des algorithmes présentés.

Un algorithme de suppression dual de l'algorithme d'insertion, doit obligatoirement associer à tout graphe $g' \in \mathcal{G}_{V+u}$, son sous-graphe induit $g'|_V$ restreint à l'ensemble V, lors de la suppression du sommet u (voir preuve de la proposition 1.7). Cette opération est bien celle réalisée par l'algorithme de suppression présenté.

La proposition 1.4 nous garantit alors directement la propriété de conservation d'aléa de l'algorithme de suppression.

Chapitre 2

Maintenance des modèles par paire

Sommaire

Ce chapitre permet d'appréhender la tâche de maintenance sur des graphes dérivés du modèle par paire, un modèle simple mais important dans le contexte de notre étude. Nous proposons des algorithmes de maintenance pour plusieurs modèles de graphes ou multigraphes basés sur le modèle par paire. Cette étude permet d'entrevoir les limites et les difficultés inhérentes à la maintenance de graphes de manière décentralisée.

Le modèle par paire (*pairing model*), aussi appelé modèle de configurations, est un modèle simple de graphes aléatoires qui a été introduit en premier par Bollobás [16] sous sa forme actuelle. Il permet de capturer certaines propriétés de plusieurs familles de graphes aléatoires (réguliers [96], sans limite d'échelle [21], etc) tout en étant facilement manipulable et simulable de manière centralisée. Nous proposons dans ce chapitre de maintenir exactement la distribution des multigraphes du

modèle par paire. Lorsque les nœuds possèdent un degré homogène pair, cette maintenance n'aura pas besoin de connaître explicitement la taille du graphe à chaque évolution du réseau, contrairement au modèle $G(n,p)$.

Nous présenterons en suivant des algorithmes de maintenance sur un modèle par attachement préférentiel, basé sur le modèle par paire. La maintenance n'est réalisée dans ce cas, que de manière semi-décentralisée lors de l'opération de suppression.

Nous terminons ce chapitre en argumentant sur la difficulté de maintenir localement directement le graphe des paires (à la place du multigraphe) et les graphes k-réguliers uniformes (un autre modèle de graphes aléatoires lié au modèle par paire).

Afin de maintenir efficacement le modèle par paire, nous allons commencer par maintenir exactement la distribution des *appariements* uniformes, correspondant aux couplages parfaits de K_V, le graphe complet sur V.

2.1 Maintenance de couplages (quasi)-parfaits

Lorsque $|V|$ est pair, un couplage parfait de K_V est une sélection de $|V|/2$ arêtes de K_V qui ne possèdent aucune extrémité en commun, *i.e.* une partition de V en $|V|/2$ paires de sommets (on parlera aussi d'*appariement*). Lorsque $|V|$ est impair, un quasi-couplage de K_V correspond à la sélection d'un unique sommet u non-apparié et d'un couplage parfait de K_{V-u}. Dans ce contexte, nous parlerons de V-couplage, ou plus simplement *couplage sur* V, pour désigner les couplages (quasi)-parfait de K_V.

Définition 2.1. Un couplage sur V est un ensemble M de $\lfloor |V|/2 \rfloor$ paires d'éléments, augmenté d'un unique singleton lorsque $|V|$ est impair, et formant une partition de V.

L'ensemble des couplages sur V est noté $\mathcal{M}_V$. L'objectif de cette section sera de préserver la distribution uniformes des couplages sur V (pour $V \subset \Omega$), qu'on appellera distribution uniforme des couplages (quasi)-parfaits.

Il n'est pas difficile d'énumérer de tels objets. Lorsque $n = |V|$ est pair, $|\mathcal{M}_V| = (n-1)!!$. Pour $u \in V$, il existe $n-1$ choix possibles pour l'élément v apparié à u. Une fois cet élément choisi, il reste à apparier $n-2$ éléments, *i.e.* choisir un couplage sur $V \setminus \{u,v\}$ où u et v sont les deux éléments déjà appariés. Nous avons donc $|\mathcal{M}_V| = (n-1) \cdot (n-3) \cdots 3 \cdot 1$.

Lorsque $n = |V|$ est impair, $|\mathcal{M}_V| = n!!$. Dans ce cas, il existe n choix possibles pour l'élément u qui ne sera pas apparié dans le couplage, et $|\mathcal{M}_{V-u}| = (n-2)!!$ choix pour coupler les éléments restants.

Ainsi, en résumé, un couplage M sur V est uniforme si
- lorsque $|V|$ est pair, M est un couplage parfait uniforme de K_V,
- lorsque $|V|$ est impair, M est un couplage quasi-parfait uniforme de K_V, *i.e.* M est formé d'un unique singleton U uniforme sur V, et conditionné à $U = u$, $M - \{u\}$ est un couplage parfait uniforme de K_{V-u},

ou de manière équivalente, M est un V-couplage uniforme si pour tout $m \in \mathcal{M}_V$, nous avons (en notant $n = |V|$) :

$$\mathbb{P}(M = m) = \begin{cases} 1/(n-1)!! & \text{si } n \text{ est pair} \\ 1/n!! & \text{si } n \text{ est impair} \end{cases}$$

Nous allons présenter des procédures afin de maintenir la distribution des couplages (quasi)-parfaits uniformes. Ces procédures seront présentées sans tenir compte de la décentralisation (*i.e.* en utilisant explicitement V) car la mise à jour se fait de manière intrinsèquement centralisée lorsque la parité du nombre de sommets change. Nous présentons ensuite des algorithmes dans notre modèle de décentralisation pour la maintenance de couplages parfaits basée sur une itération de ces procédures d'évolution.

La mise à jour des couplages sur V est basée sur une bijection explicite entre :
- pour $|V|$ pair, $\mathcal{M}_V \times (V + u)$ et $\mathcal{M}_{V+u}$;
- pour $|V|$ impair, $\mathcal{M}_V \times \{u\}$ et $\mathcal{M}_{V+u}$ avec $u \notin V$ (les deux ensembles ayant en effet la même taille).

2.1.1 Algorithme d'insertion

Nous proposons la procédure suivante afin de construire un couplage M' sur $V + u$ à partir d'un V-couplage M :

Algorithme 3 Insertion-couplages

1. Si $|V|$ est pair,
   - Tirer uniformément v dans $V + u$;
   - Si $v = u$, laisser u non-apparié, *i.e.* $M' = M + \{u\}$;
   - Sinon, soit w l'élément apparié à v dans M, remplacer la paire $\{v,w\}$ par la paire $\{w,u\}$ laissant v seul, *i.e.* $M' = M - \{v,w\} + \{w,u\} + \{v\}$;
2. Si $|V|$ est impair (M possède alors un singleton w),
   - Coupler w avec u, i.e. $M' = M - \{w\} + \{w,u\}$.

Proposition 2.1. *La procédure d'insertion décrite ci-dessus préserve la distribution uniforme des couplages (quasi)-parfaits.*

Preuve Soit M un couplage uniforme sur V, et M' le couplage obtenu après l'ajout de l'élément u au couplage M en suivant la procédure d'insertion décrite. Deux cas se présentent alors suivant la parité de $n = |V|$.

Si n est pair, M est un couplage parfait et M' possède un singleton. Pour chaque $(V + u)$-couplage m possédant u en singleton on a

$$\mathbb{P}(M' = m) = \mathbb{P}(M = m - \{u\}) \cdot 1/(n+1) = 1/|V + u|!!$$

et pour chaque $(V+u)$-couplage m possédant $v \neq u$ comme singleton et w comme élément couplé à u, on a

$$\mathbb{P}(M' = m) = \mathbb{P}(M = m + \{v,w\} - \{w,u\} - \{v\}) \cdot 1/(n+1) = 1/|V+u|!!.$$

Si n est impair, M est un couplage quasi-parfait et M' un couplage parfait. Pour tout $(V+u)$-couplage m dont v est l'élément apparié à u dans m, on a

$$\mathbb{P}(M' = m) = \mathbb{P}(M = m - \{u,v\} + \{v\}) = 1/n!! = 1/(|V+u|-1)!!.$$

□

2.1.2 Algorithme de suppression

La procédure suivante construit un couplage M' sur $V-u$ à partir d'un couplage M sur V :

Algorithme 4 SUPPRESSION-COUPLAGES

1. Si $|V|$ est pair,
   — Laisser l'élément v apparié à u dans M non-apparié dans M', *i.e.* $M' = M - \{u,v\} + \{v\}$;
2. Si $|V|$ est impair (donc M possède un singleton w),
   — Si $w = u$, retirer simplement u, *i.e.* $M' = M - \{u\}$;
   — Sinon, coupler l'élément v apparié à u dans M avec le singleton w, *i.e.* $M' = M - \{w\} - \{u,v\} + \{v,w\}$.

Proposition 2.2. *La procédure de suppression décrite ci-dessus préserve la distribution uniforme des couplages (quasi)-parfaits.*

Preuve Soit M un couplage uniforme sur V, et M' le couplage obtenu après la suppression de u du couplage M en suivant la procédure décrite. Nous avons deux cas de figure suivant la parité de $n = |V|$.

Si n est pair, pour chaque couplage m sur $V-u$ avec singleton v,

$$\mathbb{P}(M' = m) = \mathbb{P}(M = m - \{v\} + \{u,v\}) = 1/(n-1)!! = 1/|V-u|!!.$$

Si n est impair, pour chaque couplage m sur $V-u$, on obtient en posant $m_{v,w} = m + \{w\} + \{u,v\} - \{v,w\}$:

$$\begin{aligned}\mathbb{P}(M' = m) &= \mathbb{P}(M = m + \{u\}) + \sum_{\{a,b\} \in m} [\mathbb{P}(M = m_{a,b}) + \mathbb{P}(M = m_{b,a})] \\ &= 1/n!! + (n-1)/2 \cdot 2/n!! = n/n!! = 1/(|V-u|-1)!!\end{aligned}$$

□

Algorithme 5 Suppression-Couplages-Parfaits

Entrées: un couplage M sur V et deux sommets $u,v \in V$
Sortie: un couplage M' sur $V \setminus \{u,v\}$

```
1: M' ← M
2: si {u,v} ∈ M alors
3:     M' ← M' − {u,v}
4: sinon
5:     x ← N(u) ; y ← N(v)
6:     M' ← M' − {u,x} − {v,y} + {x,y}
```

2.1.3 Maintenance des couplages parfaits

Des algorithmes de mise à jour peuvent être déduits sans difficultés de la section précédente afin de maintenir la distribution des couplages parfaits uniformes. Une mise à jour pour ces couplages consiste à insérer deux nouveaux éléments n'appartenant pas à V, et une suppression à retirer deux éléments distincts du couplage courant (mais pas forcément une paire).

Pour maintenir de telles structures, il suffit alors d'exécuter deux fois séquentiellement les algorithmes présentés précédemment. Ainsi, l'insertion d'un couple de nouveaux éléments a et b dans un V-couplage M, construit dans un premier temps un couplage quasi-parfait sur $V + a$, puis insère b pour obtenir un couplage parfait sur $V \cup \{a,b\}$. On pourra noter qu'insérer b puis a fournit un couplage distribué identiquement. La suppression fonctionne de manière similaire : on supprime successivement a et b de V, afin d'obtenir un couplage parfait sur $V \setminus \{a,b\}$.

Ces algorithmes, bien que simples, possèdent une propriété intéressante dans le contexte de notre étude : ils peuvent être réalisé de manière entièrement *locale* (cf. leur description dans notre modèle, algorithmes 5 et 6). En effet, à aucun moment le couplage dans son ensemble n'a besoin d'être connu et seules des informations locales sont utilisées :

- pour la suppression de deux sommets u et v, l'enchaînement des deux suppressions revient à supprimer l'arête $\{u,v\}$ lorsqu'elle existe, ou sinon dans le cas contraire, à apparier les sommets restés seuls, *i.e.* les voisins respectifs de u et v ;
- pour l'insertion de u puis v, un sommet uniforme w est tiré (dans notre modèle, appel à `RV`) puis soit on ajoute l'arête $\{u,v\}$ (lorsque le sommet tiré est u), soit on retire l'arête $\{w,x\}$ et on ajoute les arêtes $\{u,x\}$ et $\{v,w\}$ (lorsque l'on n'a pas tiré u).

Les propositions 2.1 et 2.2 nous indiquent que le couplage obtenu après chacun des deux algorithmes présentés est bien uniforme sur les couplages parfaits.

Algorithme 6 INSERTION-COUPLAGES-PARFAITS

Entrées: un couplage M sur V et deux sommets $u,v \notin V$
Sortie: un couplage M' sur $V \cup \{u,v\}$

```
1: M' ← M
2: w ← RandomVertex(V + u)
3: si w = u alors
4:     M' ← M' + {u,v}
5: sinon
6:     x ← N(w)
7:     M' ← M' − {w,x} + {u,x} + {v,w}
```

2.2 Maintenance du multigraphe de paires

À la section précédente, nous avons proposé des algorithmes afin de maintenir des couplages parfaits. Bien qu'il soit possible de voir ces couplages comme des graphes (de degré homogène 1), ils ne forment pas de structures intéressantes. L'approche adoptée du modèle par paire est de construire un graphe à partir d'un couplage donné.

Le modèle par paire pour le degré k sur V, associe à chaque couplage M de $V \times [\![k]\!]$, un multigraphe $P(M)$ de degré k. On peut facilement déduire un graphe $G(M)$ de degré au plus k à partir du multigraphe $P(M)$, en supprimant les boucles et en fusionnant les arêtes multiples.

Ce procédé peut être utilisé pour générer un graphe k-régulier uniforme (voir e.g. [96]) en générant des multigraphes $P(M)$ à partir d'un couplage M uniforme, jusqu'à en obtenir un qui soit sans boucle ni arêtes multiples. Le graphe obtenu par ce biais est uniforme sur les graphes k-réguliers, puisque pour chaque graphe k-régulier sur V, il existe le même nombre de couplages le produisant.

Nous proposons ici de maintenir la distribution du multigraphe de paires obtenu à partir d'un couplage uniforme de $V \times [\![k]\!]$.

Multiensembles

Un multiensemble est la donnée d'un couple $E = (A,m)$ où A est l'ensemble des éléments de E et $m : A \to \mathbb{N}^*$ leur multiplicité.

On note de manière additive l'augmentation et la diminution de la multiplicité d'un élément : si $E = (A,m)$ est un multiensemble, alors $E + u$ désigne le multiensemble $(A + u,m')$ où $m'(a) = m(a)$ pour $a \in A$ et $m'(u) = m(u) + 1$ si $u \in A$, ou $m'(u) = 1$ sinon. De manière équivalente, pour $u \in A$, $E - u$ est égal à (A,m') avec $m'(u) = m(u) - 1$ si $m(u) > 1$, et $E - u = (A - u,m'')$ avec $m'' = m|_{A-u}$ sinon.

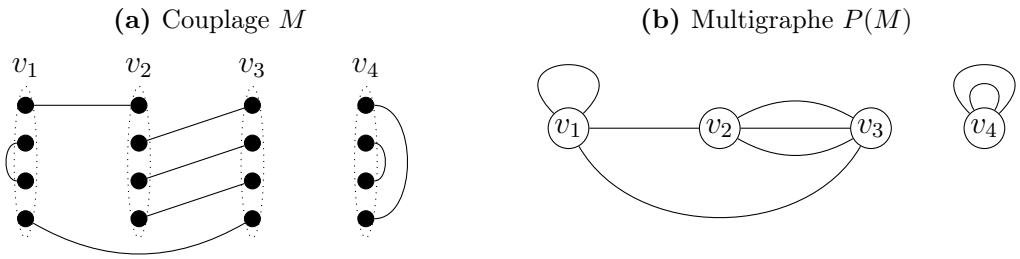


Figure 2.1 – Exemple de passage d'un couplage à un multigraphe.

2.2.1 Multigraphe de paires

Le multigraphe de paires $P(M)$ de degré k sur V est un multigraphe obtenu à partir d'un couplage M sur $V \times [\![k]\!]$. L'ensemble des sommets de $P(M)$ est V, et les arêtes de $P(M)$ sont associées aux paires $\{(u,i),(v,j)\}$ de M, pour $u,v \in V$ et $i,j \in [\![k]\!]$. Si M possède un singleton, *i.e.* si $|V|$ et k sont impairs, il n'est associé à aucune arête de $P(M)$.

Définition 2.2. Le multigraphe $P(M) = (V,E)$ est défini à partir d'un couplage M sur $V \times [\![k]\!]$ en fusionnant les éléments $((v,i))_{i \in [\![k]\!]}$ de M pour chaque $v \in V$, *i.e.* E est le multiensemble (A,m) où

— $A = \{\{u,v\} \in V^2 \mid \exists i,j \in [\![k]\!], \{(u,i),(v,j)\} \in M\}$, et
— $m(\{u,v\}) = |\{(i,j) \in [\![k]\!]^2 \mid \{(u,i),(v,j)\} \in M\}|$ pour $\{u,v\} \in A$.

L'ensemble de tels multigraphes est noté $\mathcal{P}_V^k = \bigcup_{M \in \mathcal{M}_{V \times [\![k]\!]}} P(M)$.

Il est important de noter que $P(M)$ est un multigraphe pouvant contenir des boucles, parfois appelé *pseudographe*. En effet, des boucles se forment lorsque plusieurs points appartenant au même "sommet" $v \in V$ sont sélectionnés par l'appariement M et des arêtes multiples sont présentes lorsque deux paires ou plus de la forme $((u,\cdot),(v,\cdot))$ apparaissent dans M. La figure 2.1 illustre le passage d'un couplage à un multigraphe.

Tous les sommets de $P(M)$ sont de degré k, excepté un sommet qui a degré $k-1$ lorsque $|V| \cdot k$ est impair. Notons que la seule information manquante dans $P(M)$ par rapport au couplage M est la numérotation des arêtes incidentes des sommets de V.

Définition 2.3. La distribution ρ_V^k des multigraphes de paires de degré k sur V est la distribution du multigraphe $P(M)$ lorsque M est un couplage uniforme sur $V \times [\![k]\!]$.

Remarque Si X est distribué selon la distribution ρ_V^k, alors pour tout multigraphe $G \in \mathcal{P}_V^k$,

$$\mathbb{P}(X = G) = \frac{1}{(k|V|-1)!!} \cdot |\mathcal{M}(G)|$$

où $\mathcal{M}(G)$ est l'ensemble des couplages M sur $V \times [\![k]\!]$ tels que $P(M) = G$.

Exemple 2.1. Pour $k = 2$ et $n = |V|$, il existe 2^{n-1} couplages de $V \times \{1,2\}$ donnant un cycle donné alors qu'il n'existe qu'un seul couplage aboutissant au graphe ayant n boucles. Leur probabilité respective est donc de $2^{n-1}/(2n-1)!!$ et $1/(2n-1)!!$.

On voit sur cet exemple que la distribution ρ_V^k est loin d'être uniforme : certains multigraphes sont exponentiellement plus probables que d'autres.

■

Il est possible, en réutilisant les algorithmes de mise à jour sur les couplages, de maintenir la distribution $\rho^k = (\rho_V^k)_{V \subset \Omega}$ des multigraphes de paires de degré k. Nous proposons d'expliciter ici ces algorithmes de mise à jour pour la distribution ρ^k dans le cas où k est pair.

Lorsque k est impair et $|V|$ aussi, il existe un sommet du multigraphe de paires dont le degré est $k-1$. Pour effectuer une insertion ou suppression dans ce graphe, il faut impérativement *localiser* ce sommet, car par définition $P(M)$ n'en contient pas pour la taille $(|V| \pm 1)k$. Lorsque la parité de $|V|$ est connue des algorithmes, cette localisation peut être effectuée mais aura besoin d'un temps moyen linéaire, ce qui met en échec notre politique de mise à jour efficace. Si la parité de $|V|$ n'est pas connue, la distribution ne peut être maintenue exactement car tout algorithme de maintenance ne fonctionnera pas convenablement pour $|V|k$ impair en n'effectuant pas de localisation, ou tournera infiniment pour $|V|k$ pair.

Si l'information du sommet de degré impair est centralisée, alors une adaptation des algorithmes proposés permet de mettre à jour le graphe.

2.2.2 Algorithme d'insertion

Algorithme 7 INSERTION-PAIRES

Entrées: un multigraphe $G = (V, E)$, un sommet $u \in V$
Sortie: un multigraphe G'

$V' \leftarrow V + u$
$E' \leftarrow E$
pour i de 1 à $k/2$ **faire**
    répéter
        $v \leftarrow$ **RandomVertex**(V')
        $\ell \leftarrow$ *Uniforme*$([k])$
    jusqu'à $v \neq u$ ou $\ell \leq 2i-1$
    si $v = u$ et $\ell = 2i-1$ **alors**
        $E' \leftarrow E' + \{u,u\}$
    sinon
        $x \leftarrow N_{E'}(v)[\ell]$
        $E' \leftarrow E' - \{v,x\} + \{u,v\} + \{u,x\}$
$G' \leftarrow (V', E')$

L'algorithme d'insertion consiste en la succession de $k/2$ insertions de paires de *points* (ou éléments du couplage sous-jacent au multigraphe) suivant l'algorithme d'insertion pour les couplages parfaits uniformes. Un point uniforme est obtenu en tirant un sommet uniforme sur $V + u$ et une de ses k arêtes incidentes ; lorsque le sommet u est tiré, un rejet est effectué pour garantir la distribution uniforme du point sélectionné.

Nous supposons ici que $N_E(v)[\ell]$ désigne le ℓ-ème élément du voisinage de v, pris dans un ordre *arbitrairement* choisi.

Proposition 2.3. *L'algorithme* Insertion-Paires *préserve la distribution ρ^k des multigraphes de paires de degré k.*

Preuve Soit M un couplage uniforme sur $V \times [\![k]\!]$ et $G = P(M) \sim \rho_V^k$, l'entrée de notre algorithme.

Soit M'_i le couplage de $\mathcal{M}_{V\times[\![k]\!]+u\times[\![2i]\!]}$, défini par $M'_0 = M$, et :

— $M'_i = M'_{i-1} + \{(u,2i-1),(u,2i)\}$ si (u,u) a été ajouté à E' au i-ème tour de boucle de l'algorithme (*i.e.* la ligne 9 a été exécutée) ;

— $M'_i = M'_{i-1} - \{(v,a),(x,b)\} + \{(u,2i-1),(v,a)\} + \{(u,2i),(x,b)\}$ si au contraire la ligne 12 a été exécutée au i-ème tour de boucle. Ici, les sommets v et x correspondent aux sommets utilisés dans l'algorithme au tour i (lignes 11 et 12). Quant à a et b, ils sont obtenus à partir de $N_{E'}(v)[\ell]$ (ligne 11) et M'_{i-1}, de telle sorte que la paire $p = \{(v,a),(x,b)\}$ est la j-ème paire entre v et x dans M'_{i-1}, si et seulement si, x apparaît $j-1$ fois dans la séquence $(N_{E'}(v)[i])_{1\le i<\ell}$.

La paire p existe forcément car on peut vérifier par induction sur i, que G_i, le graphe courant à la i-ème itération, est égal à $P(M'_i)$. De plus, il est important de noter qu'à une exécution complète de l'algorithme ne correspond qu'un unique couplage $M' = M'_{k/2}$.

Prouvons par récurrence que le couplage M'_i pour $0 \le i \le k/2$ est un couplage uniforme sur $V \times [\![k]\!] + u \times [\![2i]\!]$.

Par hypothèse, nous avons M'_0 est un couplage uniforme de $V \times [\![k]\!]$. Soit $m \in \mathcal{M}_{V\times[\![k]\!]+u\times[\![2i]\!]}$ et distinguons deux cas suivant si $p_u = \{(u,2i-1),(u,2i)\}$ appartient à m :

1. Si $p_u \in m$ alors
$$\mathbb{P}(M'_i = m) = \frac{\mathbb{P}(M'_{i-1} = m - p_u)}{nk + 2i - 1} = \frac{1}{(nk+2i-1)!!}$$
car la ligne 9 n'est exécutée que dans un cas sur $nk + 2i - 1$.
2. Si $p_u \notin m$, on appelle $v_a = (v,a)$ et $x_b = (x,b)$ les points appariés respectivement à $u_1 = (u,2i-1)$ et $u_2 = (u,2i)$ dans m. On a alors
$$\mathbb{P}(M'_i = m) = \frac{\mathbb{P}(M'_{i-1} = m + \{v_a,x_b\} - \{u_1,v_a\} - \{u_2,x_b\})}{nk + 2i - 1}$$
car un unique couple de valeurs pour v et ℓ supprime la paire (v_a, x_b).

Enfin comme $G' = P(M')$, nous avons G' suit la loi des multigraphes de paires de degré k sur $V + u$.

□

Le coût asymptotique de l'algorithme INSERTION-PAIRES, en nombre d'appels à la primitive `RandomVertex`(V+u), est de $k/2$ (chaque tour de boucle utilise en moyenne moins de $\frac{nk+1}{nk}$ appels), et sa complexité moyenne, en nombre de pas de calcul, est $\mathcal{O}(1)$.

Remarque Si `RandomVertex`(V+u) est simulée à partir uniquement d'un générateur sur V (e.g. `RV`), il nous faut alors en moyenne k appels asymptotiques.

2.2.3 Algorithme de suppression

L'itération de la procédure de suppression pour les couplages parfaits sur un couplage donné, ne produit pas forcément les mêmes résultats suivant l'ordre dans lequel on choisit les paires à supprimer.

Exemple 2.2. Posons $V = \{a_1,a_2,b_1,b_2,c_1,c_2,d_1,d_2\}$ et considérons le couplage parfait $M_0 = \{\{a_1,a_2\},\{b_1,b_2\},\{c_1,c_2\},\{d_1,d_2\}\}$ sur V. Il existe six ordres pour supprimer $U = \{a_1,b_1,c_1,d_1\}$, donnant trois couplages différents :

- supprimer la paire $\{a_1,b_1\}$ puis la paire $\{c_1,d_1\}$, ou dans l'ordre opposé, donne le couplage $M'_1 = \{(a_2,b_2),(c_2,d_2)\}$;
- supprimer la paire $\{a_1,c_1\}$ suivit de la paire $\{b_1,d_1\}$, ou dans l'ordre opposé, donne $M'_2 = \{(a_2,c_2),(b_2,d_2)\}$;
- enfin, la suppression de $\{a_1,d_1\}$ puis $\{b_1,c_1\}$, dans un des deux ordres possibles, produit $M'_3 = \{(a_2,d_2),(b_2,c_2)\}$.

Si on applique la procédure de suppression des couplages (quasi)-parfaits sommet par sommet, l'ordre de suppression à l'intérieur des paires n'a pas d'incidence.

■

En revanche, comme la distribution des couplages uniformes est préservée par la procédure, si M est uniforme sur $\mathcal{M}_V$, alors le couplage M' obtenu après les quatre suppressions, est uniforme sur $\mathcal{M}_{V\setminus U}$, *quelque-soit* l'ordre des suppressions (fixé a priori, indépendamment de M), découlant du principe de la préservation de lois (proposition 1.2).

Maintenant, lorsque la tâche confiée à l'algorithme est de supprimer les sommets de U mais sans préciser l'ordre de la suppression, l'algorithme peut décider d'un d'ordre arbitraire sur V, par exemple un ordre déjà connu ou un ordre uniforme.

Donc, une approche pour maintenir le multigraphe de paires, consiste à itérer la procédure de suppression dans un ordre uniforme sur les arêtes incidentes au sommet à supprimer (voir algorithme 8). Dans la description de l'algorithme, $N_E\langle u\rangle$ désigne le voisinage ouvert de u, dont les éléments sont des paires (v,i) avec $v \in V$ et $i \in [\![k]\!]$; chaque arête uv, avec $u \neq v$, de multiplicité m fournit m telles paires $(v,1),\ldots,(v,m)$. Les boucles sur u ne sont pas prises en compte ici.

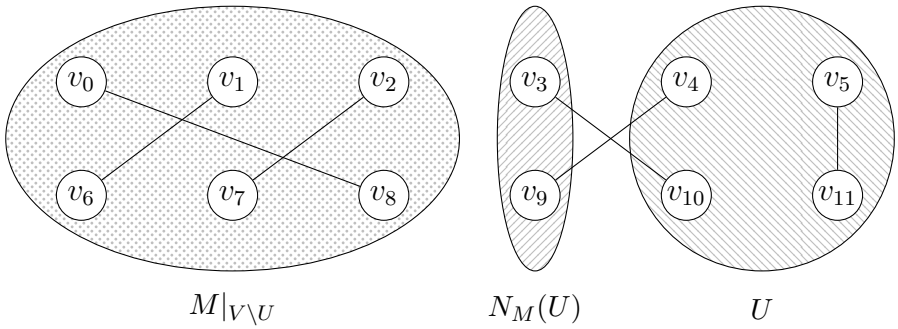


Figure 2.2 – Illustration du lemme 2.4.

Afin de prouver la correction de l'algorithme, nous introduisons un lemme sur les couplages parfaits qui se déduit de la procédure de suppression.

On note $N_U(M)$ l'ensemble des *voisins* de l'ensemble U dans le couplage M et $M|_{V\setminus U}$ le couplage entre éléments absents de $U \cup N_U(M)$ dans M, *i.e.*

— $N_U(M) = \{v \in V \setminus U \mid \exists u \in U, \{u,v\} \in M\}$, et

— $M|_{V\setminus U} = \{\{a,b\} \in M \mid a,b \notin U\}$.

Lemme 2.4. *Soient U un sous-ensemble pair de V et M un couplage uniforme sur V. Conditionné à $N_M(U) = N$, si M'' est un couplage uniforme sur N, alors $M|_{V\setminus U} + M''$ est un couplage uniforme de $V \setminus U$.*

Preuve Soit $\mathcal{S}$ notre procédure de suppression pour mettre à jour les couplages parfaits. Comme $\mathcal{S}$ ne fait pas d'appels à `RV`, l'itération de la procédure sur tous les sommets de U à partir d'un couplage parfait M, ne peut qu'appareiller des sommets de $N_U(M)$ (qui lui aussi est de taille paire), et laissera intact les sommets de $V \setminus N_U(M)$ (les couples de $M|_{V\setminus U}$).

Si M est un V-couplage uniforme et si les suppressions sont effectuées dans un ordre *indépendant* de M, alors M', le couplage obtenue après suppressions, est uniforme sur $V \setminus U$ (proposition 1.2).

Le lemme s'obtient en prenant l'ordre uniforme sur U.

Notons que les paires $\{a,b\}$ appartenant à M entre deux sommets de U n'influent en rien sur le résultat final, car une des deux situations suivantes se produit :

— soit la paire $\{a,b\}$ est sélectionnée pour être supprimée et entraîne donc aucune modification du couplage de $N_U(M)$;

— soit la paire $\{a,b\}$ fait partie d'une chaîne de ℓ paires $\{a_i,b_i\} \in M$, avec $a_i,b_i \in U$ pour tout $i \in [\![\ell]\!]$, tel que $\{u_0, u\} \in M$, $u_0 \in U, u \notin U$, u_0 et a_1 sont les premiers supprimés, $\{b_1, a_2\}$ les suivants, jusqu'à b_ℓ et v_0 avec $\{v_0, v\} \in M$, $v_0 \in U, v \notin U$. Alors en appliquant itérativement $\mathcal{S}$, on trouve qu'après la suppression finale de la paire (v_0,v) le conséquence sur le couplage de $N_U(M)$ est d'ajouter la paire $\{u,v\}$ à M'.

Donc uniquement l'ordre relatif des suppressions des sommets de U appariés à des sommets de $N_U(M)$ importe. L'ordre uniforme donne dans ce cas un couplage uniforme sur $N_U(M)$.

□

Algorithme 8 SUPPRESSION-PAIRES

Entrées: un multigraphe $G = (V, E)$, un sommet $u \in V$
Sortie: un multigraphe G'

1: $V' \leftarrow V - u$
2: $E' \leftarrow E|_{V'}$
3: $M_u \leftarrow$ Couplage uniforme sur $N_E\langle u\rangle$
4: **pour tout** $\{(v,i),(w,j)\} \in M_u$ **faire**
5: $\quad E' \leftarrow E' + (v,w)$
6: $G' \leftarrow (V', E')$

Proposition 2.5. SUPPRESSION-PAIRES *préserve la distribution ρ^k des multigraphes de paires de degré k.*

Preuve Soit M un couplage uniforme de $V \times [\![k]\!]$ et $G = P(M)$, l'entrée de notre algorithme. Soit M' le couplage obtenu en identifiant les éléments de la forme (v, j) de M_u, au *point* correspondant à la j-ème paire entre v et u dans M (en parcourant dans l'ordre de seconde coordonnée), puis en suivant le déroulement de l'algorithme. Ainsi, étant donné un certain couplage M, toute exécution de l'algorithme produira un unique couplage M'. Donc par définition $G' = P(M')$.

D'après le lemme précédent, M' est uniforme sur $\mathcal{M}_{(V-u)\times[\![k]\!]}$ d'où G' suit bien la loi visée.

□

2.3 Maintenance d'un modèle par attachement préférentiel

Nous proposons dans cette section de maintenir un modèle faisant intervenir un attachement préférentiel sur les nœuds de forts degrés. Le modèle retenu est un modèle par attachement préférentiel présenté dans [21], basé sur le modèle de croissance introduit par Barabasi et Albert [8]. Un modèle par attachement préférentiel modélise des réseaux invariants d'échelle (*scale-free*) typique du *World Wide Web*, de certains réseaux biologiques ou réseaux sociaux (cf. e.g. [7]).

Ces réseaux invariants d'échelle sont caractérisés par une distribution des degrés des nœuds suivant une loi de puissance asymptotiquement, *i.e.* la proportion $P(k)$ des nœuds ayant degré k, pour k grand, est (approximativement) proportionnelle à $k^{-\gamma}$ pour un certain paramètre γ indépendant de l'échelle du réseau.

2.3.1 Modèle de Bollobás et Riordan

Ce paragraphe est uniquement une (légère) reformulation de la description du modèle présenté par Bollobás et Riordan dans [21].

Processus de génération

Le modèle de [21] définit un processus de génération $(G_1^t)_{t\in\mathbb{N}}$ où chaque G_1^t, est un graphe aléatoire sur $[\![t]\!]$.

Dans ce processus de génération, le graphe aléatoire G_1^{t+1} est obtenu à partir du graphe précédent G_1^t en ajoutant le sommet $t+1$ et une unique arête reliant $t+1$ à un des sommets déjà présents, ou lui-même. Une extrémité de l'arête ajoutée sera ainsi $t+1$ et l'autre extrémité est choisie aléatoirement et proportionnellement au degré des sommets du graphe G_1^t ($t+1$ est considéré alors de degré 1).

Plus précisément, le processus démarre avec G_1^1, le graphe possédant un sommet isolé avec une boucle. À partir de $G_1^t = ([\![t]\!], E)$, pour $t \geq 1$, le graphe $G_1^{t+1} = ([\![t+1]\!], E + \{t+1,x\})$ est obtenu en ajoutant le sommet $t+1$ et l'arête $\{t+1,x\}$, où x est un sommet aléatoire prenant la valeur $v \in [\![t+1]\!]$ avec probabilité :

$$\mathbb{P}(x=v) = \begin{cases} \delta_{G_1^t}(v)/(2t+1) & v \in [\![t]\!] \\ 1/(2t+1) & v = t+1 \end{cases}$$

Le processus à $m > 1$ arêtes $(G_m^t)_{t\geq 0}$ engendre des graphes G_m^t sur $\{v_1, \ldots, v_m\}$. Dans ce processus, G_m^{t+1} est obtenu en ajoutant successivement m arêtes incidentes à $t+1$. L'autre extrémité des m arêtes est choisie, de manière séquentielle, proportionnellement au degré des sommets du graphe courant au moment de l'ajout (comptabilisant alors la *demi-arête* attachée à $t+1$). De manière équivalente, le graphe G_m^t est obtenu à partir de G_1^{tm}, le processus à une arête itéré tm fois, en fusionnant les nœuds $(i-1)m+1, (i-1)m+2, \ldots, im$ en un seul sommet v_i, où i appartient à $[\![t]\!]$.

Les graphes engendrés par ce processus sont des multigraphes pouvant contenir des boucles. De plus, comme les sommets sont ajoutés dans l'ordre croissant, il existe de fait une orientation naturelle des arêtes du graphe : du sommet $t+1$ au sommet choisi comme destination. Cette orientation peut être retrouvée même sans connaître les identités des sommets, en exécutant un parcours de graphe à partir des « sources », les sommets à m boucles (chaque composante connexe du graphe contient en effet un unique sommet à m boucles).

À partir d'un diagramme de cordes

Le modèle présenté au paragraphe précédent peut aussi s'étudier à partir de diagrammes de cordes (linéarisés), qui est une façon de visualiser les couplages.

Un diagramme de n cordes est un diagramme dessiné sur une ligne (orientée) contenant $2n$ points différents reliés deux-à-deux entre eux par n arcs de cercles.

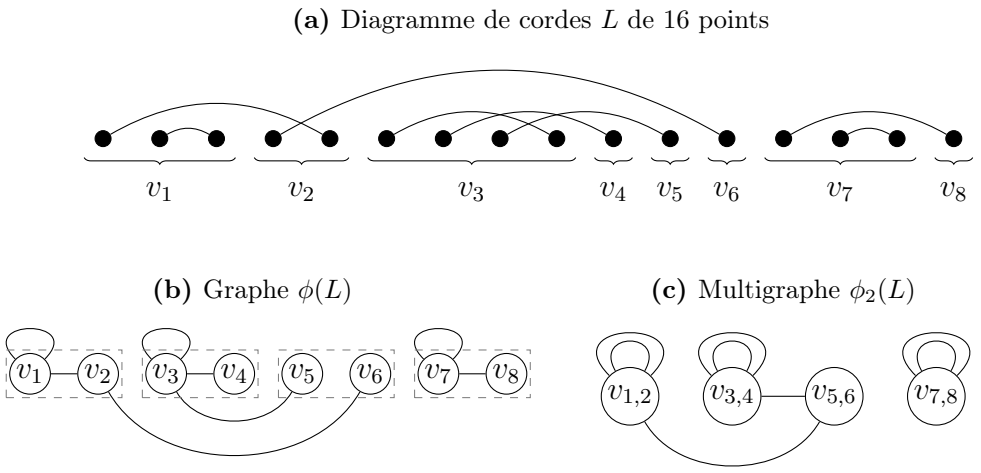


FIGURE 2.3 – Exemple de la transformation $\phi(L)$.

Nous distinguerons les deux extrémités de chacun de ces *ponts* : extrémité gauche pour le point arrivant en premier, et extrémité droite pour le deuxième.

À partir d'un diagramme de cordes L de $2n$ points, on définit un graphe $\phi(L)$ sur $\{v_1, \ldots, v_n\}$ en fusionnant les points à partir de la gauche jusqu'à la première extrémité droite pour former v_1, les suivants jusqu'à la seconde pour v_2 et ainsi de suite, et ce en conservant les *arêtes* (ponts) reliant ces points (voir figure 2.3).

Autrement dit, à la i-ème extrémité droite, on associe le sommet v_i. Pour les points compris entre les extrémités droites consécutives $i-1$ et i (ou à gauche de la première pour $i = 1$), on associe aussi le sommet v_i. Pour chaque pont reliant deux point p_i et p_j, et ajoute une arête à $\phi(L)$ reliant les sommets associés à p_i et p_j. Notons que des boucles apparaissent forcément lors du processus (exactement une par composante connexe de $\phi(L)$).

La propriété importante de cette construction est que si L est choisi uniformément parmi les diagrammes de n cordes, alors $\phi(L)$ est distribué comme G_1^n. Pour s'en rendre compte, il suffit de considérer la construction d'un diagramme de cordes L' de $n+1$ cordes uniforme à partir d'un diagramme L de n cordes uniforme ; on ajoute une corde à L en plaçant l'extrémité droite tout à droite de la ligne et en choisissant un des $2n+1$ espaces disponibles, avec équiprobabilité, pour son extrémité gauche.

L'équivalent sur le graphe $\phi(L)$, est de choisir un des sommets déjà présents avec probabilité proportionnelle à son degré, qui correspond exactement au processus défini précédemment.

Remarque Comme un diagramme de corde est juste une façon de représenter un appariement, nous pouvons voir la transformation directement à partir d'un couplage de K_{2n} (la figure 2.3a reprend par exemple le même appariement que la figure 2.1a).

Comme pour le processus de génération, la transformation ϕ s'étend pour former des graphes de degré $m > 1$, en fusionnant les extrémités droites de L, ou les sommets

de $\phi(L)$, par paquets de m (voir figure 2.3c où ϕ_2 correspond au cas $m = 2$). Le multigraphe ainsi obtenu est distribué comme G_m^n.

Notation $\phi_m(L)$ correspond au multigraphe obtenu à partir de $\phi(L)$ en fusionnant les sommets par paquet de m consécutifs.

2.3.2 Maintenance de graphes par attachement préférentiel

Le modèle G_m^n, présenté au paragraphe précédent, peut être vu comme agissant sur un ensemble V donné, en ajoutant les sommets un à un de V via le processus de génération (modèle G_m^n sur V).

Faire évoluer le graphe obtenu à la fin du processus pour V (en ajoutant ou supprimant des sommets) tout en gardant une distribution cible, ne rentre pas directement dans notre modèle de maintenance, notamment, à cause de la dépendance du modèle de l'ordre d'insertion.

Un premier problème est intrinsèquement lié au fait que la distribution du graphe construit dans le modèle G_m^n sur V est dépendante de l'ordre dans lequel les insertions sont effectuées. Un des objectifs premiers de la maintenance de graphe et de préserver une distribution quelque soit l'ordre des opérations de mise à jour ; ainsi la distribution du graphe après la séquence de mises à jour est uniquement dépendante de l'ensemble d'arrivée V. Au contraire, en suivant le processus de génération G_m^n, l'ordre des opérations d'insertions influe la distribution du graphe obtenu à la fin des mises à jour (par exemple le degré du premier sommet est en moyenne plus élevé que le dernier, de degré forcément m).

Un deuxième problème apparaît lors de la suppression. Il est facile de concevoir quelle est la distribution du graphe ciblée lorsque l'on supprime le dernier sommet ajouté, sûrement celle du graphe avant son ajout (ce qui revient à supprimer le dernier sommet et ses arêtes incidentes). Par contre, il n'est pas clair quelle serait la distribution visée lors de la suppression du premier sommet (qui possède potentiellement un degré significativement plus important que les autres).

Pour contourner ces problèmes tout en conservant une distribution de graphe aléatoire ayant le même type de propriétés, nous proposons de maintenir la distribution du modèle précédent d'attachement préférentiel G_m^n sur V, en supposant que les sommets ont été insérés via le processus de génération dans un ordre uniforme.

Définition 2.4. La distribution G_m^V est celle des graphes aléatoires $G = G_m^n[\sigma]$ où $n = |V|$, G_m^n est obtenu par le processus de génération et $\sigma : [\![n]\!] \to V$ est une bijection aléatoire uniforme entre $[\![n]\!]$ et V.

À présent, les graphes aléatoires G_m^{V-u} et G_m^{V+u} sont clairement définis.

Comme pour le modèle par paires, les graphes autorisés ici seront les multigraphes avec boucles.

Dans la suite, la famille de distributions des graphes aléatoires G_m^V, pour $V \subset \Omega$, est appelée distribution des graphes par attachement préférentiel.

Algorithme d'insertion

Afin de ne plus dépendre de l'ordre d'insertion, nous proposons de suivre la procédure habituelle lors de l'insertion d'un nouveau sommet, puis d'échanger son *voisinage* avec un sommet uniforme du graphe (lui compris). Cet échange permet d'obtenir une distribution ne dépendant plus de l'ordre dans lequel les sommets ont été ajoutés.

Définition 2.5. Échanger le voisinage de u avec celui de v dans le graphe G sur V, produit le graphe $G[\tau(u,v)]$ où $\tau(u,v)$ est la transposition $(u\ v)$, *i.e.* $\tau(u) = v$, $\tau(v) = u$ et $\tau(w) = w$ pour $w \in V \setminus \{u,v\}$.

Soit la procédure construisant le graphe $G' \in \mathcal{G}_{V+u}$ à partir de $G \in \mathcal{G}_V$ suivante :

Algorithme 9 INSERTION-ATTACHEMENT-PRÉFÉRENTIEL

1. Ajouter séquentiellement m arêtes $\{u,v_1\}$,$\{u,v_2\}$, ..., $\{u,v_m\}$ à u, où v_i est choisi par attachement préférentiel sur $V + u$ en prenant compte des arêtes ajoutées au fur et à mesure, *i.e.* pour $v \in V + u$:

$$\mathbb{P}(v_i = v) = \begin{cases} \frac{\delta_i(v)}{2m|V|+2i-1} & \text{pour } v \in V \\ \\ \frac{i+b_i(u)}{2m|V|+2i-1} & \text{pour } v = u \end{cases}$$

   où $\delta_i(v)$ est le degré du sommet v au moment de l'ajout de l'arête $\{u,v_i\}$, pour $i \in [\![m]\!]$, qui peut être calculé à partir de $\delta_G(v)$ et des sommets $v_1, \ldots, v_{i-1}$; pour u, $b_i(u)$ est le nombre de boucles déjà présentes sur u au i-ème ajout d'arêtes.

2. *Échanger* le voisinage de u avec un sommet w choisi uniformément sur $V+u$ (une boucle sur v devenant une boucle sur u et *vice versa*).

Proposition 2.6. *La procédure d'insertion décrite préserve la distribution des graphes par attachement préférentiel.*

Preuve Soient $n = |V|$, H un graphe aléatoire suivant le processus de génération G_m^n sur V et σ une bijection uniforme de $[\![n]\!]$ vers V, et $G = H[\sigma]$ l'entrée de l'algorithme. Comme la première partie de la procédure est identique au processus de génération, le graphe H' obtenu en considérant qu'on a ajouté le sommet $n+1$ (à la place de u) au graphe H est distribué selon le modèle G_m^{n+1}.

Soit σ' la bijection de $[\![n+1]\!]$ vers $V+u$ suivante : $\sigma'(i) = \sigma(v)$ pour $i \neq \sigma^{-1}(w)$ et $i \neq n+1$; $\sigma'(n+1) = w$ et $\sigma'(\sigma^{-1}(w)) = u$. Comme il est possible d'inverser la transformation (retrouver σ et $w \in V+u$), la bijection σ' est uniforme.

Nous pouvons conclure en constatant que $G' = H'[\sigma']$.

□

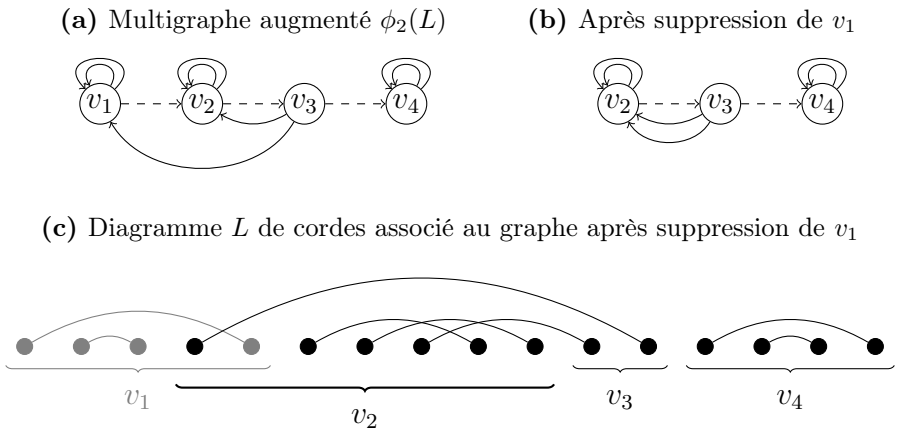


Figure 2.4 – Exemple de suppression dans $\phi_2(L)$.

Remarque Cette procédure peut être implémentée en utilisant $m + 1 + \mathcal{O}(1/n)$ appels moyen à `RV` en utilisant la méthode décrite au § 1.2.2, où la borne sur le voisinage est m ; un rejet est effectué uniquement lorsque u est tiré, avec probabilité bornée par $1 - 1/m$.

Algorithme de suppression

Nous proposons ici un algorithme de suppression qui devra utiliser de l'information supplémentaire pour garantir une maintenance exacte. D'un point de vue décentralisé, la maintenance se fera au prix d'ajouter deux artefacts sur les sommets. D'une part, nous supposons que les arêtes conservent l'orientation naturelle qui leur est donnée au moment de l'insertion (depuis le sommet inséré), et d'autre part, nous supposons qu'un arc *spécial* lie chaque sommet à celui qui a été ajouté en suivant (voir figure 2.4a, l'arc spécial est en pointillé). Ces artefacts peuvent très bien être maintenue, de manière centralisée, lors de l'insertion de nouveaux nœuds, et ce juste avant l'étape d'échange des voisinages (arcs spéciaux inclus).

Concernant l'orientation, comme indiqué précédemment, il est possible de reconstruire cette information à partir du graphe non orienté. Néanmoins, le fait de décider localement la direction d'une arête peut prendre un temps linéaire par rapport à la taille du graphe (en nombre de messages par exemple), dans le pire des cas. Un exemple nécessitant une telle complexité est de demander l'orientation du voisinage du nœud au centre dans un chemin de longueur n contenant $m-1$ boucles sur chacun des nœuds, sauf pour l'une de ces extrémités possédant alors m boucles.

Les arcs spéciaux brisent quant à eux la décentralisation du système, car au moment de l'insertion d'un nouveau sommet, la fin de la chaîne doit être localisée. Ces arcs spéciaux forment en outre un chemin orienté d'un sommet contenant m boucles

(premier « sommet » [1] ajouté) jusqu'au dernier « sommet » ajouté. Sur ce chemin, chaque sommet (sauf les extrémités), possède un successeur et un prédécesseur. Ces liens sont maintenus de sorte que lors de l'ajout d'un nouveau sommet v, l'unique sommet qui n'a pas de successeur choisit comme successeur v. Lors de l'opération d'échange de voisinage à la fin du processus d'insertion, ces liens sont aussi compris dans le voisinage.

Sous ces hypothèses, nous proposons d'utiliser la procédure suivante pour supprimer $u \in V$ à un multigraphe G distribué selon G_m^V augmenté des arcs spéciaux :

Algorithme 10 SUPPRESSION-ATTACHEMENT-PRÉFÉRENTIEL

1. Rediriger les arcs entrants du sommet u vers son successeur (destination de l'arc spécial) ; les arcs sortants de u (boucles comprises) sont simplement supprimés.

Proposition 2.7. *Le graphe obtenu (ignorant les arcs spéciaux) à la suite de la procédure de suppression est distribué selon G_m^{V-u}.*

Preuve Soient $n = |V|$, L un diagramme de cordes uniforme de $2nm$ points, $H = \phi_m(L)$ un graphe distribué selon le processus G_m^n, σ une bijection uniforme de $[\![n]\!]$ vers V, $G = H[\sigma]$ le graphe d'entrée et u le sommet à supprimer.

Notons que pour chaque sommet v de G, il correspond un ensemble $\ell(v)$ de m extrémités droites dans L. Considérons maintenant le diagramme de cordes (appariement) obtenu en supprimant les arcs de cercles dont les sommets de $\ell(u)$ forment une (ou les deux) des extrémités de l'arc. D'après la procédure de suppression dans les couplages (section 2.1.2), le diagramme de corde L' obtenu après la suppression de ces arcs de cercles est un diagramme de corde uniforme de $2(n-1)m$ points. Le graphe $H' = \phi_m(L')$ est donc distribué selon le processus G_m^{n-1}.

Maintenant, considérons la bijection $\sigma' : [\![n-1]\!] \to V-u$ obtenue à partir de σ en décrémentant les images au delà $\sigma(u)$ d'une unité, *i.e.* pour $v \in V-u$, $\sigma'(v) = \sigma(v)$ si $\sigma(v) < \sigma(u)$ et $\sigma'(v) = \sigma(v) - 1$ si $\sigma(v) > \sigma(u)$. Comme la transformation peut être inversée en choisissant $\sigma(u) \in [\![n]\!]$, σ' est une bijection uniforme de $[\![n-1]\!]$ vers $V - u$.

Il suffit de montrer à présent que G', le graphe construit après la redirection, est égal à $H'[\sigma']$. Un sommet i de H' est associé aux mêmes brins dans L' que dans L si $i < \sigma^{-1}(u)$, et aux brins de L' associés au sommet $i-1$ dans L lorsque $i > \sigma^{-1}(u)$; quant au sommet $\sigma^{-1}(u)$ de H', il est associé dans L' aux brins correspondant à $\sigma^{-1}(u)$ et à $\sigma^{-1}(u)+1$ dans L.

Par définition du successeur de u (qui correspond au sommet $\sigma^{-1}(u)+1$), on obtient $G' = H'[\sigma']$.

□

1. Ici, on différencie un « sommet » correspondant au voisinage construit à l'étape 1 de la procédure d'insertion, et l'identité de ce sommet qui est obtenu après l'étape 2.

Une suppression est présentée figure 2.4a et 2.4b, où les graphes obtenus après ajout des arcs spéciaux sont appelés multigraphes augmentés.

Une question restant ouverte est d'obtenir un algorithme de suppression totalement décentralisé pour la distribution G_m^V. L'algorithme présenté n'est pas satisfaisant pour deux raisons :

— la maintenance fait intervenir une information supplémentaire sur le multigraphe ; cette information change la topologie des communications possibles qui ne correspond plus au graphe G_m^V ;

— si cette information supplémentaire devait elle aussi être maintenue, le dernier sommet de la chaîne doit être localisé lors de l'insertion de nouveaux nœuds ; bien que celui-ci peut être récupéré dans notre modèle décentralisé en *remontant* la chaîne, ce sommet présente une centralisation du système d'un certain point de vue.

Un problème encore plus difficile serait d'effectuer la mise à jour efficacement, et ce sans utiliser aucune information supplémentaire. La stratégie actuelle de reconstruire le couplage sous-jacent suppose au minimum la connaissance de l'orientation des arcs, car cette information ne peut être acquise par des algorithmes locaux en temps sous-linéaire pour tous les graphes $\phi_m(L)$, obtenus à partir de diagrammes de cordes.

2.4 Maintenance locale difficile

Nous terminons ce chapitre en argumentant sur la difficulté de maintenir deux modèles de graphes aléatoires liés au modèle par paire. Ces deux exemples font apparaître la difficulté d'une maintenance locale efficace, alors que des algorithmes de génération satisfaisants existent pour ces deux familles de graphes aléatoires.

Le premier modèle considéré est le modèle des graphes de paires, obtenus à partir des multigraphes maintenus section 2.2, en effaçant les boucles et en fusionnant les arêtes multiples. Nous montrons sur un exemple, que le couplage sous-jacent n'est pas reconstructible localement.

Le deuxième est un modèle classique de graphes aléatoires : les graphes k-réguliers uniformes. La dépendance entre les voisinages et une génération exacte basée sur une méthode de rejet reposant sur le modèle par paire, rendent la maintenance efficace peu probable.

2.4.1 Maintenance du graphe de paires

Il est aisé de définir un graphe simple $\mathcal{G}(M)$ à partir du multigraphe $P(M)$ utilisé dans la section 2.2, en *oubliant* les boucles et *fusionnant* les arêtes multiples. Les graphes $\mathcal{G}(M)$ obtenus en prenant M uniforme se rapprochent de modèles intéressants à maintenir. En effet, les boucles et arêtes multiples ne représentent pas des liens pertinents dans les réseaux décentralisés étudiés dans cette thèse.

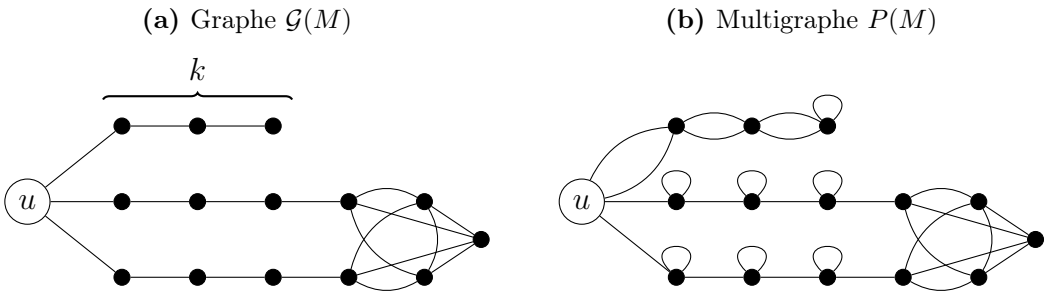


FIGURE 2.5 – Exemple de graphe et multigraphe de paires de degré 4.

Pour le cas $k = 2$, maintenir $\mathcal{G}(M)$ et $P(M)$ est équivalent car on peut facilement construire localement le multigraphe $P(M)$ à partir de $\mathcal{G}(M)$:

— un sommet de degré 0 dans $\mathcal{G}(M)$ correspond à une boucle de $P(M)$,
— un sommet de degré 1 dans $\mathcal{G}(M)$ est nécessairement associé à un autre sommet de degré 1 entre lesquels il y a une double arête,
— un sommet de degré 2 dans $\mathcal{G}(M)$ possède deux voisins distincts dans $P(M)$.

Cependant pour $k \geq 4$, le graphe de paires est plus difficilement maintenable car il n'existe pas de conditions locales afin de construire un des multigraphes $P(M)$ ayant comme graphe $\mathcal{G}(M)$, le graphe courant.

Nous proposons de montrer sur un exemple que la suppression d'un sommet u dans un graphe de paires ne peut être résolue en explorant le graphe à distance au plus k (constant) et en conservant à la fois les liens existants entre sommets distincts de u. Dans cet exemple, nous ne regarderons pas la distribution du graphe après suppression, mais uniquement s'il appartient à la *classe* des graphes de paires.

Exemple 2.3. Nous construisons le graphe G_k de degré au plus 4, illustré par la figure 2.5. Le sommet u est connecté à trois chemins A, B et C de k sommets (longueur $k-1$), et les extrémités de deux de ces chemins (par exemple B et C) sont connectées à un K_5 privé d'une arrête (voir figure 2.5a, où la partie sous l'accolade est formée d'un chemin de k sommets).

Lorsque k est impair, il n'existe qu'un unique multigraphe de paires P de degré 4, qui produit le graphe G_k en supprimant les boucles et arêtes multiples (voir figure 2.5b).

Lorsque u et ces arêtes incidentes sont supprimées, le chemin A ne contient que des sommets de degré pair (sauf ces deux extrémités) et les chemins B et C contiennent que des sommets de degré 2 sauf un pour chacun d'entre eux ; le motif entre B et C ne contient que des sommets de degré 4. Donc, pour rester dans la classe des graphes de paires (issus de multigraphes de paires), une arête doit être ajouté entre les chemins B et C, ou deux arêtes une entre le chemin A et B et l'autre entre A et C.

Enfin, si un algorithme de suppression n'explore le graphe qu'à distance bornée et ne modifie pas les liens existants, alors il ne peut décider où rajouter les arêtes

manquantes. En effet, un tel algorithme serait forcément incorrect car ne produisant pas un graphe de paires pour une des trois variantes du graphe G_k obtenues en déplaçant le chemin ne terminant pas par le motif.

■

Le précédent exemple montre que la stratégie utilisée jusqu'à présent (reconstruction d'un appariement à partir du graphe) n'est pas applicable localement pour le cas des graphes de paires.

2.4.2 Graphes k-réguliers uniformes

Un modèle de graphes aléatoires, lié au modèle par paire, est le modèle des graphes k-réguliers uniformes que nous avons déjà évoqué. Lorsque k est constant et idéalement faible, ces graphes présentent plusieurs caractéristiques recherchées lors de la construction de réseaux logiques : degrés faibles (constants), connectivité forte lorsque $k \geq 3$ (k-connexe avec forte probabilité [94, 22]) [2] et diamètre faible (logarithmique [20]) ; voir l'étude [96] de Worlmald qui recense un certain nombre de propriétés des graphes k-réguliers uniformes.

La méthode de génération de tels graphes usuellement employée lorsque k est faible est une méthode de génération par rejet basée sur le modèle par paire. Comme expliqué à la section 2.2, la méthode consiste à générer des multigraphes de paires de degré k jusqu'à en obtenir un qui soit simple. D'après [12], la probabilité d'obtenir un graphe simple est asymptotiquement de $e^{\frac{1-k^2}{4}}$. Pour des petites valeurs de k, *i.e.* $k = \mathcal{O}(\sqrt{\log(n)})$, la génération uniforme peut ainsi être réalisée efficacement (polynomialement en n en moyenne).

Un algorithme de génération exacte [95], linéaire en n dans le pire des cas, existe pour les graphes cubiques uniformes ($k = 3$ et n pair) ; un algorithme similaire peut aussi être obtenu pour couvrir les graphes 4-réguliers.

Le meilleur algorithme de génération uniforme exacte (pour k *grand*) est de complexité moyenne $\mathcal{O}(nk^3)$ lorsque $k = \mathcal{O}(\sqrt[3]{n})$, alors que des algorithmes de génération *presque* uniforme permette d'atteindre des complexités moyenne de $\mathcal{O}(nk^2)$ pour $k = \mathcal{O}(\sqrt{n})$, voir [88, 63, 11].

La maintenance exacte de graphes k-réguliers serait évidement intéressante, mais est rendue difficile par la dépendance des voisinages entre eux. Un premier problème se pose lorsque l'ensemble des sommets change de parité (pour k impair). Lorsque k est pair ($k = 4$ est alors le premier exemple intéressant), la tâche de maintenance est réalisable en ajoutant et supprimant un sommet à la fois. La complexité de l'algorithme de génération exacte [95] pour $k = 3$, et son adaptation au cas $k = 4$ (envisagée mais pas explicitée par son auteur), nous laisse pessimiste pour obtenir un algorithme de préservation simple et efficace.

2. Lorsque $k = 2$, un graphe k-régulier uniforme n'est pas connexe avec forte probabilité.

Dans le chapitre suivant, nous présenterons des algorithmes de maintenance pour les graphes k-sortants uniformes, pouvant être considérés comme une version orientée des graphes réguliers.

Puisque le degré d'un sommet est souvent proche de $2k$ dans un graphe k-sortants uniforme, ils se rapprochent des graphes $2k$-réguliers. Pour cette famille de graphes, l'indépendance des voisinages nous permettra d'obtenir des algorithmes asymptotiquement optimaux (en nombre de tirages à `RV` et en pas de calculs).

Chapitre 3

Maintenance des graphes k-sortants uniformes

Sommaire

Ce chapitre est consacré à l'étude approfondie du problème de maintenance sur un modèle simple de graphes aléatoires dirigés : les graphes k-sortants uniformes. L'étude se place dans le modèle distribué où la taille du graphe est connue lors des opérations de maintenance.

Nous présentons plusieurs algorithmes d'insertion et de suppression accomplissant la tâche de maintenance. Pour chacun d'entre eux, nous fournissons le coût

asymptotique en nombre d'appels à la primitive `RandomVertex()`, ainsi que la complexité de l'algorithme. Ils présentent tous une complexité en temps constante en moyenne, et pour les meilleurs d'entre eux, leur coût de décentralisation est asymptotiquement optimal en moyenne (parmi les algorithmes dont la complexité moyenne est bornée). Plusieurs simulations seront ensuite présentées, permettant de capturer d'autres paramètres, en dehors du coût, qui différencient ces algorithmes.

La fin du chapitre se consacre à quelques extensions possibles et modèles similaires.

3.1 Graphes k-sortants uniformes

3.1.1 Définition

Nous introduisons le modèle de graphes dirigés pour lequel nous élaborerons des algorithmes de mise à jour dans ce chapitre.

Définition 3.1. Un graphe k-sortant est un graphe dirigé simple dont tous les sommets ont comme degré sortant exactement k.

Notons dans un premier temps qu'aucune condition ne contraint les degrés entrants des sommets (qui peuvent donc varier entre 0 et $n-1$). De plus, l'adjectif *simple* rappelle que les boucles et les arêtes multiples ne sont pas autorisées contrairement aux *multigraphes* du chapitre précédent. Par contre, rien n'empêche la présence de circuits de taille 2, soit la présence simultanée des arcs (u,v) et (v,u).

Dans le reste de ce paragraphe, V est un sous-ensemble de notre univers Ω et $n = |V|$.

L'ensemble des graphes k-sortants sur V est noté $\mathcal{G}_V^k$. Par définition, $\mathcal{G}_V^k = \emptyset$ si $|V| \leq k$ car un graphe k-sortant est simple. On note $\mathcal{G}^k = (\mathcal{G}_V^k)_{V\subset\Omega,|V|\geq k+1}$.

La loi uniforme sur $\mathcal{G}_V^k$ est notée ν_V^k et $\nu^k = (\nu_V^k)_{V\subset\Omega,|V|\geq k+1}$ est la famille des distributions des graphes k-sortants uniformes sur l'univers Ω. Par définition, le comportement des algorithmes de suppression n'est pas spécifié pour les graphes k-sortants à $k+1$ sommets [1]. Un des objectifs de ce chapitre sera de maintenir efficacement la distribution uniforme en se plaçant dans notre modèle de décentralisation où les nœuds du réseau peuvent accéder à la taille exacte du réseau à mettre à jour.

Dans un graphe k-sortant de taille n, il existe $\binom{n-1}{k}$ voisinages sortants possibles pour chaque sommet du graphe. On en déduit que $|\mathcal{G}_V^k| = \binom{n-1}{k}^n$.

Un graphe dirigé aléatoire $G = (V,E)$ suit la loi ν_V^k si et seulement si :

— pour tout $v \in V$, $N_G^+(v)$ est distribué uniformément sur les k-sous-ensembles de $V - v$, et

— les $N_G^+(v)$ sont mutuellement indépendants.

1. Plus précisément, il n'existe qu'un unique graphe k-sortant à $k+1$ sommets sur V : le voisinage sortant (et entrant) de chaque sommet contient tous les autres sommets du graphe.

Cette affirmation découle directement de la définition d'un graphe k-sortant et du fait que $\mathbb{P}(G = g) = 1/\binom{n-1}{k}^n$ pour tout $g \in \mathcal{G}_V^k$.

De plus, pour u,$v \in V$ avec $u \neq v$, on a

$$\mathbb{P}(u \in N_G^+(v)) = \binom{n-2}{k-1}/\binom{n-1}{k} = k/(n-1).$$

Le degré entrant du sommet u, s'exprime comme $\sum_{v \in V-u} \mathbb{1}_{u \in N_G^+(v)}$ et les évènements $(\{u \in N_G^+(v)\})_{v \in V-u}$ sont mutuellement indépendants, on en déduit la remarque qui suit.

Remarque Si G est un graphe k-sortant uniforme, alors pour tout u dans V, le *degré entrant* de u est distribué selon la loi binomiale de paramètres $n-1$ et $k/(n-1)$, *i.e.* $\delta_G^-(u) \sim \mathrm{Bin}(n-1, \frac{k}{n-1})$.

3.1.2 Propriétés des graphes k-sortants uniformes

Les graphes k-sortants uniformes, modèle introduit par Fenner et Frieze dans [47] sous le nom de graphes m-orientables, apparaissent dans la littérature dans des contextes différents : e.g. en tant que modélisation de réseaux (sans fil, sociaux) ou pour obtenir des résultats sur les automates finis déterministes aléatoires.

Par exemple dans [52], les graphes k-sortants (appelés par attachement aléatoire) sont utilisés en comparaison de modèles par attachement préférentiel afin d'étudier la vitesse de propagation de rumeur dans les réseaux sociaux.

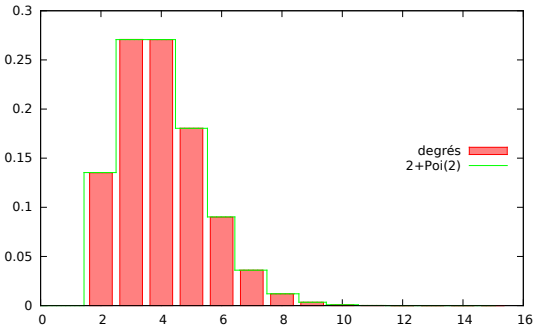
Ils apparaissent aussi récemment dans l'étude de réseaux de senseurs sans fils [97], sous un schéma de pré-distribution de clés aléatoire par paires.

Les graphes k-sortants se rapprochent aussi des automates déterministes aléatoires, à la différence que dans ces derniers des arêtes multiples sont possibles, bien que peu fréquentes. Ce modèle a récemment fait l'objet d'une étude approfondie dans [30] afin d'obtenir une génération aléatoire exacte efficace, ainsi que des analyses sur la minimisation en moyenne de tels automates (voir [9, 30]). Récemment, les graphes k-sortants ont aussi été utilisés dans le contexte d'apprentissage d'automate fini déterministe aléatoire (voir [3, 2]).

Les graphes k-sortants, associés à la distribution uniforme, possèdent des propriétés intéressantes d'un point de vue réseau. Nous rappelons ici quelques résultats montrant certaines propriétés typiquement recherché dans des réseaux, notamment de type P2P : faible degré des nœuds, forte connectivité et diamètre faible. Le paramètre k doit être ajusté afin de satisfaire les propriétés spécifiques recherchées dans un réseau donné, et déterminer sa valeur consiste à faire un compromis entre degrés des sommets et diamètre/connectivité du graphe (voir e.g. figure 3.6, table 3.2, et figure A.20 en annexe).

Comme dans notre modèle, les communications sont symétriques et les identités des nœuds n'ont pas d'importance, les propriétés sont à analyser sur le graphe à n sommets G_n^k non orienté sous-jacent. Le graphe G_n^k aléatoire contient en moyenne $nk - \mathcal{O}(k^2)$ arêtes à cause de la présence de circuits de taille 2 des graphes k-sortants se réduisant à une arête.

(a) Distribution des degrés



(b) Distribution du degré maximum

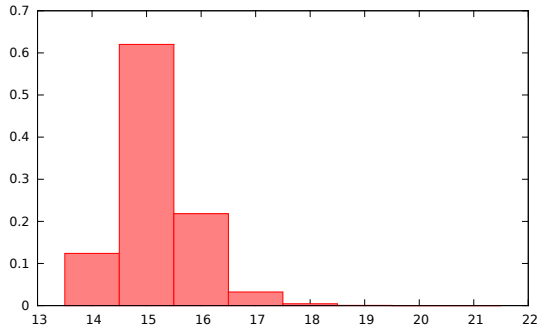


FIGURE 3.6 – Degrés dans les graphes G_n^k, pour $n = 10^7$ et $k = 2$.

Degrés

Le degré d'un sommet dans un graphe G_n^k est la somme de son degré sortant, constant à k, et de son degré entrant, qui lui suit une loi binomiale de paramètres $n-1$ et $k/(n-1)$. Les degrés possibles des sommets varient ainsi entre k et $n-1$, et le degré moyen est de $2k$.

Le degré d'un sommet particulier suit donc une loi bien connue : la probabilité que le degré prenne la valeur $j \geq k$ converge, lorsque n tend vers l'infini, vers la probabilité qu'une variable aléatoire suivant la loi de Poisson $\mathrm{Poi}(k)$ prenne la valeur $j-k$; voir figure 3.6a où sont présentes la loi $2+\mathrm{Poi}(2)$ et la distribution des degrés d'un graphe G_n^k pour 10^7 nœuds et $k = 2$ lors d'une simulation.

Cette distribution nous indique que les degrés vont être concentrés autour de $2k$ et donc que la plupart de temps, peu d'informations *locales* seront à retenir (et à manipuler) pour chaque nœud.

Un autre paramètre pouvant être considéré est le degré maximum dans un graphe G_n^k, car la présence de quelques nœuds de fort degré, même sporadiquement, entraîne un coût non négligeable dans un réseau décentralisé. Une méthode pour estimer ce degré maximum est de considérer le maximum de n $\mathrm{Poi}(k)$ indépendantes et d'y ajouter k. Les degrés entrants n'étant pas indépendants, cette démarche comporte forcément un biais. Mais en pratique, les degrés entrants étant *presque* indépendants, une estimation proche du degré maximum observé est obtenue via cette méthode.

Par exemple, l'approximation asymptotique du maximum donnée dans [24] de n $\mathrm{Poi}(k)$ indépendantes nous fournit un résultat proche du degré maximum trouvé lors de simulations de graphes k-sortants. Par exemple lorsque $n = 10^7$ et $k = 2$, l'estimation à partir de variables de Poisson nous donne un degré maximum de 15,5 alors que le degré maximum *moyen* observé est de 15,17 au cours de 10^5 simulations de graphes G_n^k indépendants ; le maximum observé des simulations est de 21 (voir figure 3.6b).

k	**1**	**2**	**3**	**4**	**5**	**8**	**16**
$\mathrm{diam}(G_n^k)$	∞ (89,7%)	9 (99,9%)	6 (64,4%)	5 (62,4%)	5	4	3
10^3 nœuds	≥ 63 (10,3%)	10 (0,1%)	7 (35,6%)	6 (37,6%)			
$\mathrm{diam}(G_n^k)$	∞ (96,1%)	11 (97,3%)	8	7	6	5	4
10^4 nœuds	≥ 231 (3,9%)	12 (2,7%)					

Table 3.2 – Exemple de diamètre observé dans les graphes G_n^k.

Connectivité et Hamiltonicité

Lorsque $k = 1$, un graphe aléatoire G_n^1 correspond à une *endofonction*[2] sans point fixe uniforme. Nous savons au moins depuis Katz [62] qu'un tel graphe n'est pas connecté avec forte probabilité.

Dans [47], Fenner et Frieze montrent en outre que, à partir du moment où $k \geq 2$, G_n^k est non seulement connexe avec forte probabilité, mais aussi k-arête-connecté et k-sommet-connecté[3].

Les mêmes auteurs montrent qu'un graphe distribué selon la loi G_n^k contient un cycle Hamiltonien[4] avec forte probabilité, tant que $k \geq 23$ (cf. [48]). Plusieurs articles ont depuis lors amélioré la borne sur k pour que récemment la borne $k = 3$ soit prouvée suffisante dans [15]. Cette borne est optimale car dans G_n^2, il existe avec forte probabilité deux sommets de degré 2 possédant un voisin commun (et donc G_n^2 n'est pas Hamiltonien avec forte probabilité).

Dans [97], les bornes connues concernant la connectivité sont améliorées.

Diamètre

Une étude précise du diamètre des graphes G_n^k n'apparaît pas, à notre connaissance, dans la littérature. Il peut être montré que pour k et ε fixe, la probabilité que le diamètre d'un graphe aléatoire G_n^k soit en $\mathcal{O}(\log(n))$ est au moins de $1 - \varepsilon$ [74].

Dans [79], le diamètre orienté des graphes k-sortants est étudié. Il est montré que lorsque $k = c\ln(n)$, avec $c > 4$, alors le diamètre d'un graphe k-sortant est a.f.p. un des deux entiers encadrant $\log_k((k-1)n+1)$. Une différence majeure avec notre travail, est de pouvoir faire dépendre k de la taille du graphe.

Enfin récemment dans [2], les auteurs montrent que le diamètre orienté des multigraphes k-sortants (chaque voisinage est tiré uniformément et indépendamment

2. fonction d'un ensemble dans lui-même.

3. Un graphe est k-sommet-connecté (resp. arrête) s'il faut au minimum enlever k sommets (resp. arrêtes) du graphe afin de le déconnecter.

4. Un cycle Hamiltonien est un cycle passant par tous les sommets du graphe. Lorsqu'un tel cycle existe, le graphe est dit Hamiltonien.

dans V^k) est $(1 + c_k + o(1)) \log_k(n)$, où c_k est constant (pour $k \geq 2$).

À titre indicatif, la table 3.2 fait apparaître le diamètre exact observé lors de 10^3 simulations de graphes G_n^k indépendants. Les pourcentages correspondent aux résultats calculés lorsqu'au moins deux résultats différents ont été obtenus.

3.2 Algorithmes de suppression

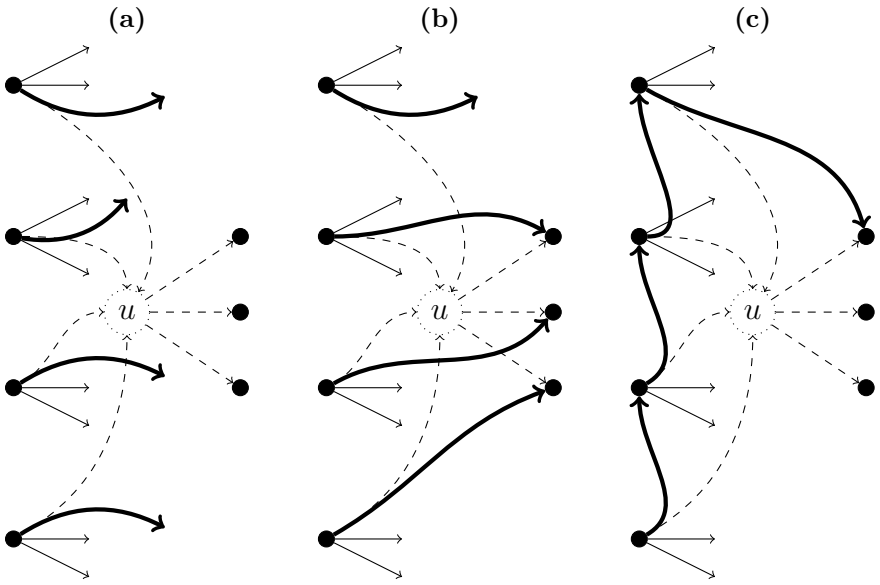


FIGURE 3.7 – Exemples de suppression de u *représentatifs* des trois algorithmes.

Lorsqu'un nœud u souhaite quitter un réseau modélisé par un graphe k-sortant, une modification des liens liant u à ses prédécesseurs est impérative pour conserver un degré sortant homogène à k pour tous les nœuds du réseau. Nous présentons dans cette section plusieurs algorithmes permettant d'une part de conserver un graphe k-sortant après l'éviction d'un sommet, et d'autre part de garantir la distribution uniforme ciblée sur ce graphe.

Après avoir supprimé le sommet u, les éventuels prédécesseurs v de u se retrouvent avec un voisinage de taille $k-1$ uniforme sur $V-u-v$. Les trois algorithmes de suppression présentés ici vont chercher, de manière de plus en plus *économique*, à transformer ces $(k-1)$-ensembles $N^+(v) - u$ aléatoires et indépendants, pour $v \in N^-(u)$, en k-ensembles aléatoires et indépendants.

Dans ce processus, uniquement les arcs liant u au reste du graphe seront supprimés, et $\delta_G^-(u)$ arcs seront ajoutés. Ces modifications représentent le nombre minimum d'arcs à enlever/ajouter afin de passer d'un graphe de $\mathcal{G}_V^k$ à un graphe de $\mathcal{G}_{V-u}^k$; en effet, lorsque $\delta_G^-(u) = \ell$, les arcs impliquant u ne peuvent être conservés ce qui force l'ajout de ℓ nouveaux arcs.

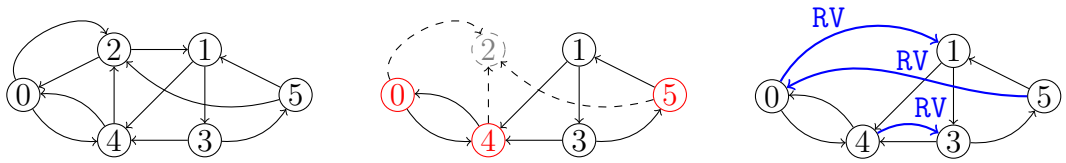


Figure 3.8 – Illustration de la suppression *naturelle* d'un sommet.

Le comportement *typique* des trois algorithmes de suppression que nous allons présenter est illustré par la figure 3.7, où le sommet u est supprimé, $k = 3$ et $|N^-(u)| = 4$: suppression naturelle 3.7a, suppression en réutilisant les successeurs 3.7b, et suppression en réutilisant les prédécesseurs 3.7c. Les arcs en pointillés sont supprimés, et les arcs en gras ajoutés ; les arcs en gras pendants sont assignés à des sommets en dehors du voisinage de u, les arcs en trait fin ne sont pas modifiés.

3.2.1 Suppression naturelle

Algorithme 11 Sup-Nat-k-sortant

Entrées: un graphe k-sortant $G = (V, E)$ et un sommet $u \in V$
Sortie: un graphe k-sortant G'

1: $V' \leftarrow V - u$
2: $E' \leftarrow E|_{V'}$
3: **pour tout** $v \in N_G^-(u)$ **faire**
4: $\quad X_v \leftarrow$ **RandomVertexAvoiding**$(N_G^+[v])$
5: $\quad E' \leftarrow E' + (v, X_v)$
6: $G' \leftarrow (V', E')$

Nous introduisons un premier algorithme Sup-Nat-k-sortant, dit de suppression naturelle, afin de supprimer un sommet u dans un graphe k-sortant tout en préservant l'uniformité du graphe : remplacer chaque arc (v,u) pointant vers u, par un arc (v,x) où x est choisi uniformément dans $V \setminus N^+[v]$ via des appels à `RandomVertex()`. Quant au sommet u et *a fortiori* ses arcs sortants, ils ne feront plus partie du graphe d'arrivée après cette suppression.

Exemple 3.1. Un exemple de suppression naturelle est donné figure 3.8 où la suppression du sommet 2 laisse 3 arêtes pendantes, qui devront être redirigées vers d'autres sommets du graphes en faisant directement appel à `RV`.

■

Proposition 3.1. Sup-Nat-k-sortant *est un algorithme de suppression préservant la distribution uniforme des graphes k-sortants.*

Preuve Nous utilisons la méthodologie suivante afin de prouver $G' \sim \nu^k_{V-u}$: pour tout $g' \in \mathcal{G}_{V-u}$ nous calculons $\mathbb{P}(G' = g')$ en supposant $G \sim \nu^k_V$ et en construisant tous les cas qui produisent g' en étudiant le déroulement de l'algorithme.

Soit $g' \in \mathcal{G}_{V-u}$ et $n = |V|$, nous partons de la formule de probabilité totale puis en analysant l'algorithme, nous obtenons

$$\begin{aligned}\mathbb{P}(G' = g') &= \sum_{g \in \mathcal{G}_V} \mathbb{P}(G' = g' \mid G = g) \cdot \mathbb{P}(G = g) \\ &= \sum_{\ell=0}^{n-1} \sum_{\substack{U \subset V' \\ |U|=\ell}} \sum_{\substack{X : U \to V' \\ X(v) \in N^+_{g'}(v)}} \sum_{\substack{Y \subset V \\ |Y|=k}} \mathbb{P}(A_{U,X} \mid G = g) \cdot \mathbb{P}(G = g)\end{aligned}$$

où $U = N^-_g(u)$ est le voisinage entrant de u dans g, $X(v) \in N^+_{g'}(v)$ correspond au nouveau voisin de v dans g' qui a remplacé u (pour $v \in U$), Y est le voisinage sortant de u dans g et $A_{U,X}$ l'évènement $\{\forall v \in U, X(v) = X_v\}$ où X_v est la variable apparaissant ligne 4 de l'algorithme ; enfin $g = g(g',U,X,Y)$ est construit de la manière suivante, pour $v \in V$:

$$N^+_g(v) = \begin{cases} Y & \text{si } v = u \\ N^+_{g'}(v) & \text{si } v \notin U \\ N^+_{g'}(v) - X(v) + u & \text{si } v \in U \end{cases}$$

Par hypothèse, on a $\mathbb{P}(G = g) = 1/\binom{n-1}{k}^n$ et on sait que les appels successifs à `RVA` sont indépendants d'où $\mathbb{P}(A_{U,X} \mid G = g) = (1/(n-1-k))^{|U|}$.

Comptant les possibilités pour les différents objets et en posant $\ell = |U|$, nous obtenons

$$\begin{aligned}\mathbb{P}(G' = g') &= \sum_{\ell=0}^{n-1} \binom{n-1}{\ell} \cdot k^\ell \cdot \binom{n-1}{k} \cdot \frac{1}{(n-1-k)^\ell} \cdot \frac{1}{\binom{n-1}{k}^n} \\ &= \frac{1}{\binom{n-1}{k}^{n-1}} \sum_{\ell=0}^{n-1} \binom{n-1}{\ell} \cdot \left(\frac{k}{n-1-k}\right)^\ell \\ &= \frac{1}{\binom{n-1}{k}^{n-1}} \left(1 + \frac{k}{n-1-k}\right)^{n-1} \\ &= \left(\frac{\binom{n-1}{n-1-k}}{\binom{n-1}{k}}\right)^{n-1} = \frac{1}{\binom{n-2}{k}^{n-1}}\end{aligned}$$

□

Proposition 3.2. *Le coût moyen asymptotique sur $|V|$, en nombre d'appels à `RV`, de* SUP-NAT-K-SORTANT *est k.*

Preuve Pour chacun des prédécesseurs de u, un appel à `RVA` est effectué, faisant en moyenne $\frac{n-1}{n-1-k}$ appels à la primitive `RV`.

Comme dans G, le sommet u possède en moyenne k prédécesseurs, au total le nombre moyen d'appels est donc de $k\frac{n-1}{n-1-k} = k + \mathcal{O}(1/n)$. □

Remarque La complexité moyenne de Sup-Nat-k-sortant (en pas de calculs) est $\mathcal{O}(1)$, en considérant k comme constant. Plus précisément, la complexité moyenne est en $\mathcal{O}(k^2)$ en considérant le pire cas pour n, soit $n = k + 1$.

Ce premier algorithme fournit une solution simple et peu coûteuse. Néanmoins, comme dans notre situation, nous souhaitons minimiser l'aléa global, nous allons présenter des algorithmes qui abaissent le nombre moyen d'appels à la primitive globale `RV`, et ce tout en conservant une complexité constante en moyenne.

3.2.2 Suppression en réutilisant les successeurs

Algorithme 12 Sup-Succ-k-sortant

Entrées: un graphe k-sortant $G = (V, E)$, et un sommet $u \in V$
Sortie: un graphe k-sortant G'

1: $V' \leftarrow V - u$
2: $E' \leftarrow E|_{V'}$
3: $S \leftarrow \emptyset$
4: **pour tout** $v \in N_G^-(u)$ **faire**
5: **si** $|S| < k$ **alors**
6: $X \leftarrow \text{Uniforme}(N_G^+(u) \setminus S)$
7: **si** $\text{Bernoulli}(|S|/|V'|)$ **alors**
8: $X_v \leftarrow \text{Uniforme}(S)$
9: **sinon**
10: $X_v \leftarrow X$
11: $S \leftarrow S + X$
12: **si** $X_v \in N_G^+[v]$ **alors**
13: $R_v \leftarrow$ **RandomVertexAvoiding**$(N_G^+[v])$
14: **sinon**
15: $R_v \leftarrow X_v$
16: **sinon**
17: $R_v \leftarrow$ **RandomVertexAvoiding**$(N_G^+[v])$
18: $E' \leftarrow E' + (v, R_v)$
19: $G' \leftarrow (V', E')$

Nous décrivons ici Sup-Succ-k-sortant, un deuxième algorithme effectuant la suppression d'un sommet dans un graphe k-sortant, qui améliore le coût de décentralisation (nombres d'appels à `RV`). L'idée principale est de *réutiliser* le voisinage

sortant du sommet à supprimer u dans le but d'économiser des appels à la primitive ; on rappelle que $N^+(u)$ est un sous-ensemble de $V-u$ de taille k, choisi aléatoirement et uniformément ; de plus, il est indépendant des autres voisinages sortants.

Le i-ème voisin sortant de u va être proposé généralement au i-ème prédécesseur (pour les k premiers) ; afin de *briser* la dépendance entre les sommets constituant $N^+(u)$, les $i-1$ premiers successeurs pourront être choisis avec probabilité $1/(n-1)$ chacun. Pour les prédécesseurs de u au delà du k-ème, s'il y en a, et lorsque la proposition de remplacement ne convient pas, la primitive `RV` est appelée comme pour l'algorithme précédent.

Proposition 3.3. SUP-SUCC-K-SORTANT *est un algorithme de suppression préservant la distribution uniforme des graphes k-sortants.*

Afin de prouver l'affirmation précédente, nous introduisons le lemme suivant portant sur les X_v (lignes 8 et 10 de l'algorithme) ; la preuve de la préservation de la loi ν^k suit.

Lemme 3.4. *Conditionné à ce que $u_1, \ldots, u_k$ soient les k premiers prédécesseurs de u (avec $u_i = \perp$ si $i > \delta^-(u)$), les $(X_{u_i})_{i\in[\![k]\!],u_i\neq\perp}$ sont des variables aléatoires i.i.d. uniformes sur V'.*

Preuve Soit S_i la valeur de S au début du i-ème tour de boucle ; S_i est donc un ensemble de taille $i-1$. Soit $\ell = \min\{k, \delta^-(u)\}$.

Calculons $P_i(v,a,s) = \mathbb{P}(X_{u_i} = v \mid X_{u_1} \ldots X_{u_{i-1}} = a_1 \ldots a_{i-1}, S_i = s)$ par induction sur $1 \leq i < \ell$, où $v \in V'$, et $a,s \in (V')^{i-1}$.

Deux situations distinctes se présentent alors :

— Si $v \in s$, alors $P_i(v,a,s) = \frac{i-1}{|V'|} \cdot \frac{1}{i-1} = \frac{1}{|V'|}$. Ce cas correspond à un succès du générateur de Bernoulli (ligne 7), puis à la sélection de v à la ligne 8 par l'appel à `Uniforme`(S).

— Si $v \notin s$, alors $P_i(v,a,s) = (1 - \frac{i-1}{|V'|}) \cdot \frac{1}{|V'|-(i-1)} = \frac{1}{|V'|}$. Ici, le générateur de Bernoulli a échoué et on remarque que conditionné à $s \subset N_G^+(u)$, X (ligne 6) est uniforme sur $V' \setminus s$.

Dans les deux cas, la probabilité conditionnelle est $1/|V'|$.

Par induction sur i, la probabilité que le vecteur $(X_{u_i})_{i\in[\![\ell]\!]}$ soit égal à un vecteur quelconque est $\left(\frac{1}{|V'|}\right)^\ell$.

□

Preuve Soit $G = (V, E) \sim \nu_V^k$ et $u \in V$ les entrées de l'algorithme.

Nous prouvons que conditionné à $N^-(u) = N$ avec $N \subset V'$, G' suit la distribution $\nu_{V'}^k$ et ce quelque soit N. Pour prouver cela, nous montrons que, sous la condition $N^-(u) = N$, les $(N_{G'}^+(v))_{v\in V'}$ sont mutuellement indépendants, et que chaque voisinage $N_{G'}^+(v)$ suit la distribution uniforme sur les k-sous-ensembles de $V' - v$.

Nous appelons $U \subset N$, l'ensemble des (au maximum) k premiers prédécesseurs de u dans l'ordre de la boucle, ligne 4 de l'algorithme. D'après le lemme précédent, $(X_v)_{v\in U}$ est un vecteur d'i.i.d uniformes sur V'. Nous complétons ce vecteur, pour $v \in V' \setminus U$, afin de couvrir tous les v, de telle sorte que les $(X_v)_{v\in V'}$ forment un vecteur d'i.i.d uniformes sur V'.

Supposons, pour les besoins de la modélisation du fonctionnement de l'algorithme, que nous ayons à disposition un vecteur infini $(R_{i,v})_{i\geq 1, v\in V}$ de variables aléatoires indépendantes, où $R_{i,v}$ représente le résultat du i-ème appel à `RV` au tour de boucle concernant v. On note $I_v = \inf\{i : R_{i,v} \notin N_G^+[v]\}$. Pour le tour de boucle impliquant le prédécesseur v, le sommet retourné par `RVA` lignes 13 et 17 est alors modélisé par $R_{I_v,v}$.

D'après l'algorithme, nous obtenons pour $v \in V'$ que

$$N_{G'}^+(v) = \begin{cases} N_G^+(v) & \text{si } u \notin N_G^+(v) \\ N_G^+(v) - u + X_v & \text{si } v \in U \text{ et } X_v \notin N_G^+[v] \\ N_G^+(v) - u + R_{I_v,v} & \text{sinon} \end{cases}$$

En remarquant que $v \in U$ est déterministiquement décidé à partir de N et de l'ordre de la boucle *pour* de l'algorithme, qui n'est pas aléatoire, on obtient que $N_{G'}^+(v)$ est une fonction déterministe du triplet $P_v = \langle N_G^+(v), X_v, (R_{i,v})_{i\geq 1}\rangle$.

Conditionnellement à N, comme les $(P_v)_{v\in V'}$ sont indépendants, les $(N_{G'}^+(v))_{v\in V'}$ le sont aussi.

Prouvons à présent l'uniformité de $N_{G'}^+(v)$ pour chaque v. Conditionnellement à $u \notin N_G^+(v)$, nous avons $N_{G'}^+(v) = N_G^+(v)$, et c'est bien un k-sous-ensemble uniforme de $V - u$ comme voulu. Conditionnellement à l'évènement complémentaire $u \in N_G^+(v)$, $N_{G'}^+(v)$ est obtenu à partir de $N_G^+(v)$ en enlevant u (donnant un $(k-1)$-sous-ensemble de $V - u$ uniforme) puis en ajoutant un sommet uniforme de $V - u$, conditionné à ne pas être dans $N_G^+[v]$; le résultat est donc bien un k-sous-ensemble uniforme de $V - u$.

Nous avons ainsi prouvé que, pour tout ensemble $N \subset V - u$, conditionné à $N_G^-(u) = N$, le graphe G' est uniforme sur $\mathcal{G}_{V-u}$. Ceci est donc également vrai sans conditionnement.

□

Proposition 3.5. *Le coût moyen asymptotique sur $|V|$ de* SUP-SUCC-K-SORTANT *est* $\frac{e^{-k}k^k}{(k-1)!}$.

Preuve Notons $n = |V'|$ et $L = |N^-(u)|$. Pour $i \in \mathbb{N}$, soit B_i une variable de Bernoulli valant 1 si une des lignes 13 ou 17 est exécutée au cours du i-ème tour de boucle et 0 sinon (en particulier, lorsque la boucle fait moins de i tours).

Remarquons que B_i vaut 1 avec probabilité $\mathbb{P}(L \geq i)\frac{k}{n}$ pour $i \leq k$ et avec probabilité $\mathbb{P}(L \geq i)$ pour $i > k$ (parce que $|S|$ augmente de 1 à chaque passage par les lignes 6-11).

Soit $\mathcal{R} = \sum_{i\geq 1} B_i$; comme `RVA` dans l'algorithme utilise $\frac{n}{n-k}$ tirages à `RV` en moyenne, le coût moyen de l'algorithme est simplement $\mathbb{E}(\mathcal{R})\frac{n}{n-k} = \mathbb{E}(\mathcal{R})(1 + \mathcal{O}(1/n))$. Par ailleurs

$$\mathbb{E}(\mathcal{R}) = \sum_{i=1}^{n} \mathbb{E}(B_i) = \frac{k}{n}\sum_{i=1}^{k} \mathbb{P}(L \geq i) + \sum_{i=k+1}^{n} \mathbb{P}(L \geq i).$$

Le premier terme est $\mathcal{O}(1/n)$ car $\sum_{i=1}^{k} \mathbb{P}(L \geq i) \leq k$. On réécrit le second terme avec quelques manipulations simples de sommes :

$$\begin{aligned}
\sum_{i=k+1}^{n} \mathbb{P}(L \geq i) &= \sum_{i=k+1}^{n}\sum_{j=i}^{n} \mathbb{P}(L = i) = \sum_{i=k+1}^{n} (i-k)\mathbb{P}(L = i) \\
&= \sum_{i=k+1}^{n} i\mathbb{P}(L = i) - \sum_{i=k+1}^{n} k\mathbb{P}(L = i) \\
&= k - \sum_{i=0}^{k} i\mathbb{P}(L = i) - k\left(1 - \sum_{i=0}^{k} \mathbb{P}(L = i)\right) \\
&= k\sum_{i=0}^{k} \mathbb{P}(L = i) - \sum_{i=1}^{k} i\mathbb{P}(L = i).
\end{aligned}$$

Or comme L suit la loi binomiale $\mathrm{Bin}(n, k)$, nous avons

$$\mathbb{P}(L = i) = \binom{n}{i}\frac{k^i(n-k)^{n-i}}{n^n} = \frac{n-i+1}{i}\cdot\frac{k}{n}\cdot\frac{n}{n-k}\cdot\mathbb{P}(L = i-1)$$

pour $i \geq 1$, d'où

$$\sum_{i=1}^{k} i\mathbb{P}(L = i) = \sum_{j=0}^{k-1} k\cdot\frac{n-j}{n-k}\mathbb{P}(L = j) = k\sum_{j=0}^{k-1} \mathbb{P}(L = j) + \mathcal{O}(1/n).$$

Additionnant les deux termes, on obtient $\mathbb{E}(\mathcal{R}) = k\mathbb{P}(L = k) + \mathcal{O}(1/n)$, et il suffit de noter que $\lim_{n\to\infty} \mathbb{P}(L = k) = k^k/(k!e^k)$ pour conclure.

□

Remarque La formule d'approximation de Stirling $n! \geq \sqrt{2\pi n}(n/e)^n$, nous indique que le coût moyen asymptotique est inférieur à $k/\sqrt{2\pi k} = \sqrt{\frac{k}{2\pi}}$.

La complexité moyenne en temps reste bornée par $\mathcal{O}(k^2)$ pour cet algorithme.

Algorithme 13 Sup-Pred-k-sortant

Entrées: un graphe k-sortant $G = (V, E)$, et un sommet $u \in V$
Sortie: un graphe k-sortant G'

$V' \leftarrow V - u$
$E' \leftarrow E|_{V'}$
$u_1, ..., u_L \leftarrow \text{Permuter}(N_G^-(u))$
pour $1 \leq i \leq L$ **faire**
    si $i = L$ **alors**
        $X \leftarrow \text{Uniforme}(N_G^+(u))$
        si $X \in N_G^+[u_L]$ **alors**
            $X \leftarrow$ **RandomVertexAvoiding**$(N_G^+[u_L])$
    sinon
        $C \leftarrow \{u_j \mid j < i \text{ and } u_j \notin N_G^+(u_i)\}$
        si $\text{Bernoulli}(|C|/(|V'| - k))$ **alors**
            $X \leftarrow \text{Uniforme}(C)$
        sinon si $(u_i, u_{i+1}) \notin E'$ **alors**
            $X \leftarrow u_{i+1}$
        sinon
            $X \leftarrow$ **RandomVertexAvoiding**$(N_G^+[u_i] \cup C)$
    $E' \leftarrow E' + (u_i, X)$
$G' \leftarrow (V', E')$

3.2.3 Suppression en réutilisant les prédécesseurs et successeurs

Nous introduisons dans ce paragraphe un dernier algorithme de suppression pour les graphes k-sortants, dont le coût asymptotique est optimal. Dans le dernier algorithme, la raison pour que le coût soit d'environ $\sqrt{k}$ est la possibilité que u puisse avoir plus de k prédécesseurs. Ici, nous allons utiliser des sommets qui sont aussi nombreux que les prédécesseurs : les prédécesseurs eux-mêmes. Ainsi, alors que l'algorithme précédent utilisait judicieusement les successeurs du sommet à supprimer pour économiser des appels à `RV`, l'idée du dernier algorithme est d'utiliser ses propres prédécesseurs comme *propositions* pour le remplacer, plus un unique successeur. De manière un peu surprenante, il est possible de *corriger* ces propositions pour conserver l'indépendance.

Plus précisément, l'algorithme propose au i-ème prédécesseur u_i du sommet supprimé u un sommet comme remplacement de u, dans l'ordre qui suit :

— un des sommets parmi $u_1 \dots u_{i-1}$ pouvant remplacer u dans le voisinage de u_i ; chacun de ces *candidats remplaçants* est choisi avec probabilité $1/(|V'| - k)$;
— le sommet u_{i+1} s'il convient (c'est-à-dire si $u_{i+1} \notin N_G^+(u_i)$) ;
— un sommet choisi uniformément parmi $V' \setminus (N_G^+[u_i] \cup \{u_1 \dots u_{i-1}\})$.

Proposition 3.6. SUP-PRED-K-SORTANT *est un algorithme de suppression préservant la distribution uniforme des graphes k-sortants.*

Preuve Nous démontrons que les $(N^+_{G'}(v))_{v\in V'}$ sont indépendants, et que pour chaque $v \in V'$, $N^+_{G'}(v)$ est uniforme sur les k-sous-ensembles de $V' - v$, en supposant que $G \sim \nu^k_V$ soit l'entrée de l'algorithme. On pose $|V| = n$.

Nous avons déjà l'uniformité pour les $v \in V \setminus N^-_G(u)$: conditionnellement à $u \notin N^+_G(v)$, $N^+_G(v)$ est bien uniforme sur les k-sous-ensembles de $V' - v$.

Soit P, l'ensemble des prédécesseurs (dans G) de u. Conditionnellement à P, les ensembles $(N^+_G(v))_{v\in P}$, sont indépendants et uniformes sur les k-sous-ensembles de $V - v$ contenant obligatoirement u ; c'est-à-dire $N^+_G(v) = N_v + u$, où les N_v sont des sous-ensembles de $V' - v$ indépendants, de taille $k - 1$ et uniformes. Donc (conditionnellement à P, et à la collection $(N_v)_{v\in P}$), il suffit pour obtenir un graphe G' suivant la distribution ν_k, d'obtenir une collection $(x_v)_{v\in P}$ de sommets aléatoires indépendants, avec x_v uniforme sur $V' - v - N_v$, et de prendre $N^+_{G'} = N_v + x_v$. Cette tâche est effectuée par la boucle de l'algorithme : la valeur de X à la fin du i-ème tour de boucle est x_{u_i}. La suite de la preuve consiste à prouver cette affirmation, par conditionnements successifs.

Conditionnellement à $L = |P|$, u_1 est un sommet uniformément distribué sur V'. De plus, conditionnellement à u_1 et N_{u_1}, x_1 doit être uniforme sur $V' - u_1 - N_1$. Or, sous ces conditions, u_2 est uniforme sur $V' - u_1$, et l'algorithme le conserve comme x_1 s'il appartient à l'ensemble plus petit $V' - u_1 - N_1$, sinon il utilise `RVA` pour obtenir x_1 avec la distribution appropriée.

Pour les $i < L$ suivants, conditionnellement à la séquence u_1,N_1,x_1,u_2,..., x_{i-1}, u_i, l'ensemble N_i doit être un $(k-1)$-sous-ensemble uniforme de $V' - u_i$: ce qui est bien le cas en utilisant $N^+_G(u_i) - u$. Alors, conditionnellement à N_i, x_i doit être uniforme sur $V' - u_i - N_i$. Autrement dit, x_i doit être un élément de $U_{i-1} = \{u_1, \ldots, u_{i-1}\}$ avec probabilité $c/(n - 1 - k)$ où c est le nombre de sommets parmi U_{i-1} qui n'appartiennent pas à N_i ; cette partie est assurée par les lignes 10-12 dans l'algorithme. Avec probabilité complémentaire, x_i doit être un sommet uniforme de $V' - u_i - N_i - U_{i-1}$. Sous ces conditions, u_{i+1} est un sommet uniforme de l'ensemble plus grand $V' - \{u_1, \ldots, u_i\}$, et donc le tirage avec rejets (lignes 13-16) donne bien la distribution voulue pour x_i.

Enfin, en ce qui concerne le dernier cas $i = L$, on sélectionne un sommet uniforme de $N^+_G(u)$, qui est indépendant de tout ce qui a été examiné jusqu'à présent, et on conserve ce sommet comme x_L s'il n'appartient pas à N_L ; dans le cas contraire, `RVA` est utilisé.

□

Proposition 3.7. *Le coût moyen de* SUP-PRED-K-SORTANT *est* $\mathcal{O}(1/n)$.

Preuve Nous travaillerons en conditionnant sur la valeur ℓ de L, le nombre de prédécesseurs de u dans G. Notons que la probabilité que L soit plus grande que, disons, $(n-1)/2 - k$ est exponentiellement faible en n.

De plus, conditionnellement à une aussi grande valeur pour L, l'espérance du nombre d'appels à `RV` réalisés dans tout l'algorithme, à partir des appels à `RVA`, est bornée par n^2 (au maximum n appels à `RV` en moyenne pour chaque appel à `RVA`, et au maximum n appels à `RVA` pour tout l'algorithme). Donc, la contribution au coût moyen d'une valeur aussi anormalement grande de L est exponentiellement faible.

Nous supposons à présent que $L < (n-1)/2 - k$. Il existe L appels possibles à `RVA` (un pour chaque tour de boucle). Pour $i < L$, le i-ème appel possible (*i.e.* au i-ème tour de boucle) ne se produit pas notamment si $u_{i+1} \notin N_G^+(u_i)$, c'est-à-dire, la probabilité que `RVA` est effectivement appelée est bornée par $(k-1)/(n-2)$.

Comme à la ligne 16, l'ensemble à éviter est de taille bornée par $\ell + k$, le i-ème appel à `RVA` contribue au pire à hauteur de $\frac{k-1}{n-2} \cdot \frac{n-1}{n-\ell-k} \leq 2 \cdot \frac{k-1}{n-2}$ au coût moyen total.

De manière similaire, l'appel potentiel à `RVA` du L-ième tour de boucle n'est effectué qu'avec probabilité $k/(n-1)$, et comme `RVA` possède un ensemble à éviter de taille k, le coût moyen du L-ème tour de boucle correspond à $\frac{k}{n-1} \cdot \frac{n}{n-k} = \mathcal{O}(1/n)$ appels à la primitive `RV`.

En résumé, en prenant l'espérance de L, tout en gardant à l'esprit que $\mathbb{E}(L|L < n/2) \leq \mathbb{E}(L) = k$, le nombre total d'appels à `RV` est borné par

$$\frac{2(k-1)^2}{n-2} + \frac{nk}{(n-1)(n-k)}$$

ce qui, pour n'importe quelle constante k, est $\mathcal{O}(1/n)$.

□

Remarque La complexité moyenne reste bornée lorsque k est constant.

Ce dernier algorithme peut être encore amélioré en réutilisant plus de successeurs de u. On peut, par exemple, réutiliser la technique utilisée dans l'algorithme Sup-Succ-k-sortant afin de produire un vecteur de k variables indépendantes et uniformes sur $V - u$. Chacune de ces *propositions* pourra être utilisée avant d'appeler la première fois la fonction `RVA`, lorsque ce *tampon* de propositions sera vidé. Même si évidement le coût asymptotique ne peut être amélioré, les constantes cachées dans le coût moyen en $\mathcal{O}(1/n)$ seront plus faibles pour ce dernier algorithme, ce qui ne changera de manière notable le coût que pour les petites valeurs de n.

3.3 Algorithmes d'insertion

Nous allons présenter à présent des algorithmes d'insertion préservant la distribution des graphes k-sortants uniformes. Le comportement « typique » des trois différents algorithmes d'insertion pour u est illustré figure 3.9 avec $k = 3$ et $|N^-(u)| = 4$: l'insertion naturelle 3.9a, l'insertion en réutilisant les successeurs 3.9b et l'insertion en réutilisant les prédécesseurs 3.9c. Les conventions utilisées dans la figure sont les mêmes que pour la figure 3.7.

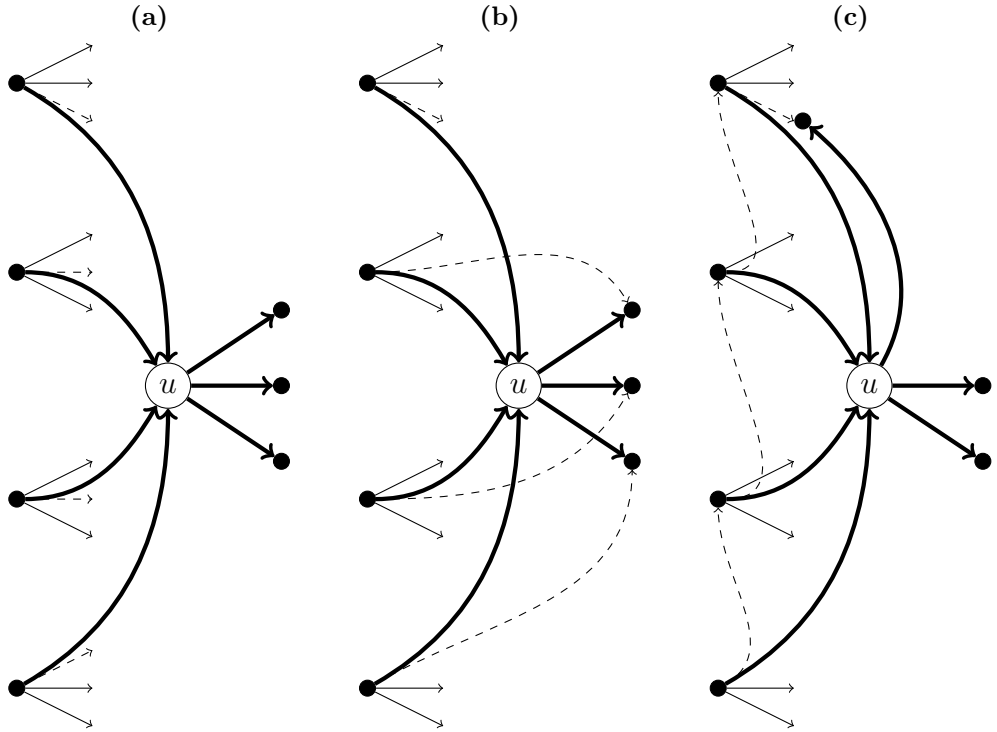


FIGURE 3.9 – Exemples d'insertion de u *représentatifs* des trois algorithmes.

3.3.1 Insertion naturelle

Nous introduisons un premier algorithme, dit d'insertion naturelle, afin d'insérer un sommet dans un graphe k-sortant uniforme tout en préservant la distribution souhaitée.

Dans un premier temps, un sous-ensemble uniforme de taille k de l'ensemble des sommets V du graphe est sélectionné comme voisinage sortant de u directement via la fonction `RandomSubset`, donc en faisant des appels successifs à `RV`.

Remarquons que le graphe produit après avoir ajouté les arcs entre le nouveau sommet u et le voisinage sortant qui lui a été attribué, bien qu'étant un graphe k-sortant, ne suit pas la loi uniforme : le nouveau sommet possède forcément un degré entrant de zéro alors que celui-ci est nulle avec probabilité $(1 - k/n)^n \xrightarrow{n} e^{-k}$ dans un graphe k-sortant uniforme sur $V + u$; le graphe obtenu suit la loi d'un graphe k-sortant uniforme sur $V + u$, conditionné à ce que u soit de degré entrant nul.

Dans un second temps, l'algorithme va construire un ensemble de prédécesseurs pour u. Pour ce faire, il va choisir en premier lieu la taille L du voisinage entrant de u en tirant $L \sim \mathrm{Bin}(n, k/n)$. Puis, l'algorithme sélectionne un ensemble W de taille L, uniforme sur les L-sous-ensembles de V, via la fonction `RandomSubset`. Enfin, pour chaque sommet v ayant été sélectionné comme voisin entrant de u, un arc sortant uniforme de v est choisi pour être *redirigé* vers u.

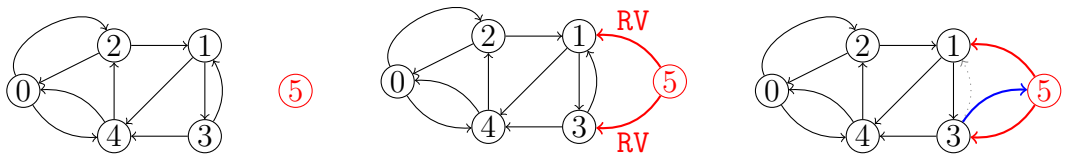


FIGURE 3.10 – Illustration de l'insertion *naturelle* d'un sommet.

Autrement dit, cette dernière étape consiste à choisir un sommet x_v uniforme sur $N_G^+(v)$, puis enlever l'arc (v,x_v) et ajouter l'arc (v,u), et ceci pour chaque $v \in W$.

Bien que simple, cet algorithme préserve bien la distribution uniforme des graphes k-sortants.

Exemple 3.2. Une exécution de l'algorithme d'insertion naturelle est donnée figure 3.10. Afin d'ajouter le sommet 5, deux sommets distincts (1 et 3) lui sont choisis comme successeurs, à l'aide de `RV`. On choisit ensuite $X \sim \mathrm{Bin}(5, 2/5)$ sommets aléatoires distincts comme prédécesseurs, et *vole* une arrête de chacun d'entre eux (ici $X = 1$, et l'ensemble $N^-(5) = \{3\}$ est choisi). ■

Algorithme 14 INS-NAT-K-SORTANT

Entrées: un graphe k-sortant $G = (V, E)$, et un sommet $u \notin V$
Sortie: un graphe k-sortant G'

1: $V' \leftarrow V + u$
2: $S \leftarrow$ **RandomSubset**(k)
3: $E' \leftarrow E \cup \{(u, v) \mid v \in S\}$
4: $L \leftarrow \mathit{Binomiale}(|V|, k/|V|)$
5: $W \leftarrow$ **RandomSubset**(L)
6: **pour tout** $v \in W$ **faire**
7: $\quad X_v \leftarrow \mathit{Uniforme}(N_G^+(v))$
8: $\quad E' \leftarrow E' - (v, X_v) + (v, u)$
9: $G' \leftarrow (V', E')$

Proposition 3.8. INS-NAT-K-SORTANT *est un algorithme d'insertion préservant la distribution uniforme des graphes k-sortants.*

Preuve Afin de prouver $G' \sim \nu_{V+u}^k$, nous calculons $\mathbb{P}(G' = g')$ pour tout $g' \in \mathcal{G}_{V+u}$ en supposant $G \sim \nu_V^k$ et en construisant tous les cas produisant g' en étudiant le déroulement de l'algorithme.

Soit $g' \in \mathcal{G}_{V+u}$, en posant $\ell = |N_{g'}^-(u)|$, nous avons

$$\mathbb{P}(G' = g') = \sum_{\substack{X : N_{g'}^-(u) \to V, \\ X(v) \notin N_{g'}^-[v]}} \mathbb{P}(A_{g',X} \mid G = g) \cdot \mathbb{P}(G = g) \tag{3.3}$$

où $A_{g',X}$ est l'évènement $\{N^+_{G'}(u) = N^+_{g'}(u)$ et $N^-_{G'}(u) = N^-_{g'}(u)\}$, et $g = g(g',X)$ est construit de la manière suivante, pour $v \in V$:

$$N^+_g(v) = \begin{cases} N^+_{g'}(v) & \text{si } u \notin N^+_{g'}(v) \\ N^+_{g'}(v) - u + X(v) & \text{si } u \in N^+_{g'}(v) \end{cases}$$

Par hypothèse, on a $\mathbb{P}(G = g) = 1/\binom{n-1}{k}^n$. Sous l'évènement $G = g$, $A_{g',X}$ s'exprime comme

$$\{S = N^+_{g'}(u)\} \cap \{L = \ell\} \cap \{W = N^-_{g'}(u)\} \cap \{X_v = [N^+_g(v) \Delta N^+_{g'}(v)] - u\}$$

où S, L, W et X_v sont les variables utilisés dans l'algorithme. Par conséquent, le terme de l'équation 3.3 ne dépend pas de X. Le nombre de ces termes est $(n-k)^\ell$. Par définition de `RandomSubset`, `Binomiale` et `Uniforme`, les différents évènements de $A_{g',X}$ sont indépendants, et on obtient

$$\begin{aligned} \mathbb{P}(G' = g') &= (n-k)^\ell \frac{1}{\binom{n}{k}} \binom{n}{\ell} (k/n)^\ell ((n-k)/n)^{n-\ell} \frac{1}{\binom{n}{\ell}} \frac{1}{k^\ell} \frac{1}{\binom{n-1}{k}^n} \\ &= \left(\frac{n-k}{n}\right)^n \frac{1}{\binom{n}{k}} \frac{1}{\binom{n-1}{k}^n} = \frac{1}{\binom{n}{k}^{n+1}} \end{aligned}$$

□

Proposition 3.9. *Le coût moyen asymptotique sur $|V|$ de* INS-NAT-K-SORTANT *est de $2k$.*

Preuve On fait appel à `RV` uniquement lors des deux appels à `RandomSubset` : une première fois avec paramètre k et une seconde avec L, où $\mathbb{E}(L) = k$. Chacun de ces appels utilise asymptotiquement en moyenne k appels à `RV`.

□

Remarque La complexité moyenne est encore une fois en $\mathcal{O}(1)$ en considérant k constant. Le pire cas se produit lorsque n est proche de k, et correspond encore une fois à une complexité de $\mathcal{O}(k^2)$.

3.3.2 Insertion en réutilisant les successeurs

De manière similaire à l'algorithme de suppression, présenté §3.2.2, il est possible d'insérer le nouveau sommet u en économisant les tirages qui auraient normalement été utilisés pour construire les successeurs de u. Pour cela, l'algorithme procède exactement comme l'algorithme précédent pour la construction des prédécesseurs.

Concernant les successeurs, pour $i \leq k$, la destination de l'arc redirigé du i-ème prédécesseur est réutilisée afin de fournir une proposition comme i-ème successeur à u. Si cette proposition ne peut être conservée (ce sommet fait déjà partie des

Algorithme 15 Ins-Succ-k-sortant

Entrées: un graphe k-sortant $G = (V, E)$, et un sommet $u \notin V$
Sortie: un graphe k-sortant G'

$V' \leftarrow V + u$
$E' \leftarrow E$
$S \leftarrow \emptyset$
$L \leftarrow \mathit{Binomiale}(|V|, k/|V|)$
$W \leftarrow \textbf{RandomSubset}(L)$
pour tout $v \in W$ **faire**
    $X \leftarrow \mathit{Uniforme}(N_G^+(v))$
    $E' \leftarrow E' - (v, X) + (v, u)$
    si $|S| < k$ **alors**
        $C \leftarrow \{w \in N_G^+[v] - X \mid w \notin S\}$
        si $\mathit{Bernoulli}(|C|/(|V| - |S|))$ **alors**
            $S \leftarrow S + \mathit{Uniforme}(C)$
        sinon si $X \notin S$ **alors**
            $S \leftarrow S + X$
        sinon
            $S \leftarrow S + \textbf{RandomVertexAvoiding}(S \cup C)$
$S' \leftarrow \textbf{RandomSubsetAvoiding}(k - |S|, S)$
$E' \leftarrow E \cup \{(u, v) \mid v \in S \cup S'\}$
$G' \leftarrow (V', E')$

successeurs de u), on appelle `RVA` pour obtenir un sommet sur le bon ensemble. Une probabilité est aussi donnée aux autres voisins du i-ème prédécesseur afin de *corriger* la dépendance qui existe au sein du voisinage. Dans le cas où il y aurait moins de k prédécesseurs, la fonction `RandomSubsetAvoiding` est appelée afin de compléter les successeurs de u.

Proposition 3.10. Ins-Succ-k-sortant *est un algorithme d'insertion préservant la distribution uniforme des graphes k-sortants.*

Preuve Soient $G = (V, E) \sim \nu_V^k$ et $u \notin V$ les entrées de l'algorithme. Comme pour les preuves précédentes, notre but afin de montrer que $G' \sim \nu_{V'}^k$, est de prouver l'indépendance des $(N_{G'}^+(v))_{v \in V'}$, et que chaque $N_{G'}^+(v)$ est distribué uniformément sur les k-sous-ensembles de $V + u - v$.

Considérons tout d'abord W, l'ensemble des prédécesseurs de u dans G'. D'après l'algorithme, W est construit en tirant $L \sim \mathrm{Bin}(n, k/n)$ puis en tirant L sommets distincts de l'ensemble V. Une manière équivalente est de considérer L comme la somme de n Bernoulli indépendantes $(B_v)_{v \in V}$ de paramètre k/n, qui serviront à décider si $u \in N_{G'}^+(v)$. Nous considérons ici ce dernier cas.

Soit X_v le sommet retourné par `Uniforme`$(N^+_G(v))$ à la ligne 7 (X dans l'algorithme) ou $X_v = \perp$ si $B_v = 0$. Nous obtenons la relation suivante pour $v \in V$:

$$N^+_{G'}(v) = \begin{cases} N^+_G(v) & \text{si } B_v = 0 \\ N^+_G(v) + u - X_v & \text{si } B_v = 1 \end{cases}$$

Les $(N^+_{G'}(v)_v)_{v \in V}$ sont indépendants car les triplets $\langle N^+_G(v), B_v, X_v \rangle$ sont indépendants : chacun est construit déterministiquement à partir de $N^+_G(v)$, B_v et les bits aléatoires utilisés par `Uniforme`, lesquels sont tous indépendants.

Vérifions maintenant que les $N^+_{G'}(v)$ sont bien distribués. Commençons par $v \neq u$. Soient $v \in V'$, $N_v \subset V' - v$ et considérons les deux cas complémentaires $u \notin N_v$ et $u \in N_v$:

— $\mathbf{u \notin N_v}$: $\mathbb{P}(N^+_{G'}(v) = N_v \text{ et } B_v = 0) = 1/\binom{n-1}{k} \cdot \frac{n-k}{n} = 1/\binom{n}{k}$

— $\mathbf{u \in N_v}$: $\mathbb{P}(N^+_{G'}(v) = N_v \text{ et } B_v = 1) = 1/\binom{n-1}{k-1} \cdot \frac{k}{n} = 1/\binom{n}{k}$.

Dans chacun des deux cas, en remarquant que les probabilités calculées sont nulles lorsque l'on complémente B_v, nous obtenons que chaque $N^+_{G'}(v)$ est bien distribué uniformément.

Enfin, nous pouvons calculer la distribution de $N^+_{G'}(u)$ en conditionnant sur les premiers sommets formant le voisinage sortant de u, de manière similaire à ce qui est effectué dans la preuve de la proposition 3.6. Conditionnellement à tous les autres voisinages $(N^+_{G'}(v))_{v \in V}$, on a $\ell = |N^-_{G'}(u)|$ et les $j = \min(\ell,k)$ premiers sommets $v_1, \ldots, v_j$ (dans l'ordre d'exécution de la boucle *pour* de l'algorithme) de $N^+_{G'}(u)$ sont obtenus ainsi (on note u_i le i-ème prédécesseur de u) :

— v_1 est uniforme sur V par définition de $N^+_G(u_1)$, précisément $v_1 = x$ avec probabilité $k/n \cdot 1/k = 1/n$ pour chaque sommet $x \in N^+_{G'}(u_1) - u$ et avec probabilité $(1 - k/n)1/(n-k) = 1/n$ pour chaque sommet x en dehors de $N^+_{G'}(u_1) - u$;

— conditionnellement à $v_1, \ldots, v_{i-1}$, v_i est uniforme sur $V \setminus \{v_1, \ldots, v_{i-1}\}$: chaque sommet de $N^+_{G'}(u_i) - u$ est présent avec probabilité $1/(n-i+1)$, de même pour les sommets en dehors de $N^+_{G'}(u_i) - u$ (par définition de $N^+_{G'}(u_i)$ et à cause du rejet ligne 13).

Les $k - j$ derniers sommets formant le voisinage sortant de u sont obtenus via `RandomSubsetAvoiding`, ce qui par définition nous garantit qu'ils ont effectivement la distribution souhaitée.

□

Proposition 3.11. *Le coût moyen asymptotique sur $|V|$ de* INS-SUCC-K-SORTANT *est de $k(1 + e^{-k}k^k/k!) \leq k + \sqrt{\frac{k}{2\pi}}$.*

Preuve La preuve est une adaptation de celle de la proposition 3.5 : l'espérance du nombre d'appels à `RVA` est

$$k + \sum_{i=k+1}^{n} \mathbb{P}(L \geq i)$$

car lorsque L est inférieur ou égal à k, la contribution à l'espérance asymptotique du coût est k (L appels en moyenne pour construire W, $\mathcal{O}(1/n)$ appels au cours des L tours de boucles, et $k-L$ appels pour construire S') ; on fera L appels en moyenne lorsque $L > k$ (L appels pour construire W, $\mathcal{O}(1/n)$ appels au cours des L tours de boucles, et 0 appel à la ligne 17).

□

3.3.3 Insertion en réutilisant les prédécesseurs

Nous introduisons maintenant un dernier algorithme d'insertion, possédant un coût moyen de k lorsque et donc strictement *meilleur* que ceux présentés aux paragraphes précédents. Ce coût est d'ailleurs optimal (voir proposition 3.14) si on se limite à examiner des algorithmes dont la complexité moyenne est constante (ce qui est le cas de tous les algorithmes présentés jusqu'à présent). L'idée sous-jacente de ce dernier algorithme va être de construire dans un premier temps les prédécesseurs du nouveau sommet u en utilisant en moyenne un seul appel à `RV` au total (quand n est grand), puis ses successeurs en utilisant $k-1$ appels.

Plus précisément, l'algorithme commence par tirer le nombre de prédécesseurs de u : $L \sim \mathrm{Bin}(n, k/n)$ avec $n = |V|$. Puis, il utilise `RV` afin de choisir le premier prédécesseur u_1 de u. Ensuite, pour $1 \leq i < L$, une fois le i-ème prédécesseur choisi, plusieurs actions seront effectuées :

1. un arc sortant uniforme de u_i est redirigé vers u, le sommet d_i destination de l'arc avant redirection est alors sauvegardé ;
2. on choisit comme prédécesseur suivant u_{i+1} un sommet dans l'ordre qui suit :
   (a) parmi les voisins sortants de u_i n'appartenant pas à $W_{i-1} = \{u_1, \ldots, u_{i-1}\}$, chacun est choisi comme prochain prédécesseur avec probabilité $1/(n-i)$;
   (b) d_i est sélectionné comme prochain prédécesseur, sauf si $d_i \in W_{i-1}$;
   (c) u_{i+1} est sélectionné à l'aide de `RVA`, en évitant les prédécesseurs déjà sélectionnés et le voisinage sortant de u_i.

Lorsque le L-ème prédécesseur est choisi, on effectue uniquement l'action 1 décrite ci-dessus (redirection d'un arc de u_L vers u).

Le premier successeur de u est alors choisi :
— soit directement via `RV`, dans le cas où $L = 0$;
— soit en choisissant un voisin sortant de u_L et lui compris, différent de d_L, avec probabilité $1/n$ chacun ;
— soit enfin en sélectionnant d_L.

Le reste des successeurs de u sont sélectionnés via `RandomSubsetAvoiding`, en évitant le premier déjà sélectionné.

Proposition 3.12. Ins-Pred-k-sortant *est un algorithme d'insertion préservant la distribution uniforme des graphes k-sortants.*

Algorithme 16 INS-PRED-K-SORTANT

Entrées: un graphe k-sortant $G = (V, E)$, et un sommet $u \notin V$
Sortie: un graphe k-sortant G'

1: $V' \leftarrow V + u$
2: $E' \leftarrow E$
3: $L \leftarrow \mathit{Binomiale}(|V|, k/|V|)$
4: $X \leftarrow \textbf{RandomVertex}()$
5: $W \leftarrow \emptyset$
6: **pour** $1 \leq i \leq L$ **faire**
7: $\quad W \leftarrow W + X$
8: $\quad D \leftarrow \mathit{Uniforme}(N_G^+(X))$
9: $\quad E' \leftarrow E' - (X, D) + (X, u)$
10: $\quad$ **si** $i < L$ **alors**
11: $\quad\quad Y \leftarrow \{v \in N_G^+(X) - D \mid v \notin W\}$
12: $\quad\quad$ **si** $\mathit{Bernoulli}(|Y|/(|V| - i))$ **alors**
13: $\quad\quad\quad X \leftarrow \mathit{Uniforme}(Y)$
14: $\quad\quad$ **sinon**
15: $\quad\quad\quad X \leftarrow D$
16: $\quad\quad\quad$ **si** $X \in W$ **alors**
17: $\quad\quad\quad\quad X \leftarrow \textbf{RandomVertexAvoiding}(W \cup N_G^+(X))$
18: $\quad$ **sinon si** $\mathit{Bernoulli}(k/|V|)$ **alors**
19: $\quad\quad X \leftarrow \mathit{Uniforme}(N_G^+[X] - D)$
20: $\quad$ **sinon**
21: $\quad\quad X \leftarrow D$
22: $S \leftarrow \textbf{RandomSubsetAvoiding}(k - 1, \{X\})$
23: $E' \leftarrow E' + (u, X) \cup \{(u, v) \mid v \in S + X\}$
24: $G' \leftarrow (V', E')$

Preuve Remarquons tout d'abord que L est choisi ligne 3 en suivant la loi binomiale de paramètres n et k/n (avec $n = |V|$), ce qui correspond bien à la distribution du nombre de prédécesseurs de u dans un graphe k-sortant uniforme sur $V + u$.

Maintenant, conditionnellement à $L = \ell$ pour $0 \leq \ell \leq n$, la distribution cible $\nu^k_{V'}$, où en plus les prédécesseurs de u sont ordonnés dans un ordre uniforme, est caractérisée par les conditions suivantes :

— l'ensemble ordonné $N^-_{G'}(u)$ est uniforme sur tous les ℓ-sous-ensembles ordonnés de V ;
— conditionnellement à $N^-_{G'}(u) = N$ (pour un ℓ-sous-ensemble N de V), les ensembles $N^+_{G'}(v)$, pour chaque $v \in V$, sont indépendants, et chaque $N^+_{G'}(v)$ suit la distribution uniforme sur les k-sous-ensembles de $V - v$ (si $v \notin N$) ou la distribution uniforme sur les $(k-1)$-sous-ensembles de $V - v$, augmenté de u, si $v \in N$.

Cette description ne correspond pas dans l'immédiat à la façon dont l'algorithme d'insertion sélectionne les prédécesseurs de u et leur voisinage respectif. Nous donnons une autre description équivalente de la distribution cible, mais cette fois-ci plus proche de la construction de l'algorithme : conditionnellement à $L = \ell$, la distribution visée est caractérisée par :

— u_1 est uniforme sur V ;
— pour $i < L$ entier, conditionnellement à la séquence u_1, $N^+_{G'}(u_1)$, u_2, …, $N^+_{G'}(u_{i-1})$, u_i : le sommet u_{i+1} et l'ensemble $N^+_{G'}(u_i)$ sont indépendants, u_{i+1} est uniforme sur $V - \{u_1, \ldots, u_i\}$, et $N^+_{G'}(u_i) - u$ est uniforme sur les $(k-1)$-sous-ensembles de $V - u_i$;
— conditionnellement à la séquence u_1, $N^+_{G'}(u_1)$, …, $u_{\ell-1}$, $N^+_{G'(u_{\ell-1})}$, u_ℓ : l'ensemble $N^+_{G'}(u_\ell) - u$ est uniforme sur les $(k-1)$-sous-ensembles de $V - u_\ell$;
— conditionnellement à la séquence en entière u_1, $N^+(u_1)$, …, u_ℓ, $N^+_{G'(u_\ell)}$, tous les autres $N^+_{G'}(v)$ sont indépendants, chacun uniforme sur les k-sous-ensembles de $V - v$ (ou V, pour $v = u$).

Cette formulation suit l'ordre dans lequel l'algorithme présenté choisit les différents sommets. Afin de prouver la proposition, il nous suffit alors de vérifier les conditions une après l'autre.

La première condition est évidente : u_1 est obtenu via `RV` à la ligne 4 de l'algorithme.

La seconde condition est satisfaite au i-ème tour de boucle (au début de la boucle, la valeur de X est u_i). Au début du tour, $N^+_G(u_i)$ est indépendant de toutes les variables aléatoires utilisés précédemment, et il est uniforme sur les sous-ensembles de $V - u_i$ de taille k. Les lignes 8 et 9 de l'algorithme construisent $N^+_{G'}(u_i)$ en enlevant un élément uniforme de $N^+_G(u_i)$, ce qui produit bien un $(k-1)$-sous-ensemble de $V - u_i$, avant d'y ajouter u. À ce moment de l'algorithme, D est l'élément supprimé de $N^+_G(u_i)$, et conditionnellement à $N^+_{G'}(u_i)$, D est uniforme sur $V \setminus N^+_{G'}[u_i]$. Les lignes 11 à 17 de l'algorithme assurent que, conditionnellement à $N^+_{G'}(u_i)$, la valeur de X (qui deviendra u_{i+1} au prochain tour de boucle) est effectivement uniforme sur $V \setminus \{u_1, \ldots, u_i\}$: la probabilité qu'un sommet choisi uniformément dans cet ensemble de cardinalité $n-i$ appartienne à $N^+_{G'}(u_i)$ est $|Y|/(n-i)$, avec $Y = N^+_{G'}(u_i) \cap (V \setminus W)$, ce qui correspond bien à la probabilité de choisir X dans Y (ligne 12) ; avec probabilité complémentaire, X est choisi dans l'ensemble complémentaire, *i.e.* $V \setminus (W \cup N^+_{G'}(u_i)))$, en *acceptant* D s'il convient (rappel : D est uniforme sur l'ensemble plus grand $V \setminus N^+_{G'}[u_i]$), et sinon en appelant `RVA`.

La troisième condition correspond au cas $i = \ell$ dans l'algorithme, et est similaire ; ici X est utilisé pour sélectionner le premier élément de $N^+_{G'}(u)$ uniformément sur V, sans avoir recours à `RV`, soit en choisissant un élément de $N^+_{G'}[u_\ell] - u$ (avec probabilité k/n), ou en prenant D, qui sous nos conditions, est uniforme dans l'ensemble complémentaire. L'ensemble $N^+_{G'}(u)$ est complété en ajoutant $k-1$ sommets dans $V - X$, cette fois-ci en utilisant `RVA`. Notons, qu'un cas spécial se produit lorsque $\ell = 0$, le premier sommet de $N^+_{G'}(u)$ étant alors directement obtenu via un appel à `RV`.

Enfin, la quatrième et dernière condition est satisfaite en réutilisant simplement les ensembles $N_G^+(v)$ comme $N_{G'}^+(v)$ pour tous les autres sommets.

□

Proposition 3.13. *Le coût asymptotique sur $|V|$ de* INS-PRED-K-SORTANT *est k.*

Preuve Soit $\mathcal{R}$ le nombre de fois que la ligne 17 est exécutée pendant le déroulement de l'algorithme. Exception faite de cette ligne, la primitive `RV` est appelée à la ligne 4 et lors de l'appel à la fonction `RandomSubsetAvoiding` à la ligne 22. Notons que l'espérance totale est de $k + \mathbb{E}(\mathcal{R})\,(1 + \mathcal{O}(1/n))$.

Pour $i \in \mathbb{N}$, soit B_i une variable de Bernoulli égale à 1 si la ligne 17 est exécutée durant le i-ème tour de boucle, et 0 sinon (en particulier, lorsque la boucle se termine avant) ; donc, $B_i = 1$ avec probabilité bornée par $\frac{i}{n-(k-1)}\mathbb{P}(L \geq i)$. Nous avons que $\mathcal{R} = \sum_i B_i$, ce qui correspond en espérance à

$$\mathbb{E}(\mathcal{R}) \leq \sum_{i=1}^{n} \frac{i}{n-k}\mathbb{P}(L \geq i) = \frac{1}{n-k}\sum_{i=1}^{n} i\mathbb{P}(L \geq i).$$

Or par une simple inversion de somme, on obtient

$$\sum_{i=1}^{n} i\mathbb{P}(L \geq i) = \sum_{i=1}^{n}\sum_{j=i}^{n} i\mathbb{P}(L = j) = \sum_{j=1}^{n} \frac{j(j+1)}{2}\mathbb{P}(L = j) = \mathbb{E}\left(\frac{L^2+L}{2}\right).$$

Comme $L \sim \mathrm{Bin}(n, k/n)$, l'espérance de L est k et sa variance est $k(1-k/n)$, et donc $\mathbb{E}\left(L^2+L\right)/2 \leq k^2/2 + k = \mathcal{O}(1)$.

Revenant à $\mathcal{R}$, nous obtenons $\mathbb{E}(\mathcal{R}) = \mathcal{O}(1/n)$.

□

Optimalité

En conclusion de cette section, nous montrons l'optimalité de INS-PRED-K-SORTANT (en ce qui concerne son coût asymptotique) parmi les algorithmes d'insertion préservant ν^k et de complexité moyenne constante.

Rappelons que le nombre de graphes k-sortants sur un ensemble de taille n est $\binom{n-1}{k}^n$. En calculant le rapport $r_n = |\mathcal{G}_{V+u}|/|\mathcal{G}_V|$, nous obtenons qu'il y a $\binom{n}{k}(n/(n-k))^n$ fois plus de graphes sur $V+u$ que sur V (c'est-à-dire avec un sommet de plus).

Ce rapport nous apprend que pour produire un graphe k-sortant de taille $n+1$, à partir d'un graphe k-sortant de taille n, il est nécessaire d'obtenir $\log_2(r_n) = k\log_2(n) + \mathcal{O}(1)$ bits d'information supplémentaires. Chaque appel à `RV` fournit $\log_2(n)$ bits d'information, car son résultat est uniforme sur un ensemble de taille n.

Si `RV` était l'unique source d'aléa des algorithmes de mise à jour que nous considérons, alors l'affirmation précédente nous garantirait qu'il est nécessaire d'appeler en moyenne au minimum k fois la primitive.

Ceci n'est pas notre cas, et en réalité, un seul appel (asymptotiquement) à `RV` est suffisant en explorant la composante connexe [5] du sommet renvoyé par `RV`, puis en vérifiant via une comparaison à $|V|$ si tous les sommets ont été trouvés :

— si c'est bien le cas, on exécute un des algorithmes d'insertion en simulant les prochains appels à `RV` en tirant *en interne* des uniformes sur V,

— si ce n'est pas le cas, on exécute un des algorithmes d'insertion comme présenté dans cette section.

Comme les graphes k-sortants sont connexes avec probabilité asymptotiquement 1, un seul appel sera réalisé avec probabilité proche de 1 lorsque n est grand.

Bien que performant du point de vue du nombre d'appels à `RV`, un tel algorithme met en échec notre politique de maintenance efficace : sa complexité est, en moyenne, linéaire dans la taille du graphe, alors qu'elle était en moyenne constante pour les algorithmes présentés jusqu'à présent.

Si on tient compte uniquement des algorithmes dont la complexité moyenne est constante, alors Ins-Pred-k-sortant est asymptotiquement optimal.

Proposition 3.14. *Le coût asymptotique, en nombre d'appels à `RV`, de n'importe quel algorithme d'insertion préservant ν^k, de complexité moyenne constante, est en moyenne au minimum de k.*

Preuve Soit $\mathcal{A}$ un algorithme préservant ν^k et de complexité moyenne bornée. Soit K le nombre d'appels à `RV` effectués par $\mathcal{A}$ au cours d'une exécution sur un graphe sur V, où $n = |V|$.

Supposons que $\mathcal{A}$ effectue en moyenne strictement moins de k appels à `RV`, *i.e.* $\mathbb{E}(K) \leq k(1-\varepsilon)$ pour un certain $\varepsilon > 0$, donc $\mathbb{P}(K \leq k-1) \geq \varepsilon$.

Considérons une exécution de $\mathcal{A}$ sur $g \in \mathcal{G}_V$ et u, utilisant c tirages de bits uniformes (donc faisant au moins c pas de calculs), r appels à `RV` et produisant un certain graphe $g' \in \mathcal{G}_{V+u}$. Alors, la probabilité d'obtenir g' est au moins égale à

$$\left(\frac{1}{2}\right)^c \left(\frac{1}{n}\right)^r \frac{1}{|\mathcal{G}_V|}.$$

Si $r < k$, alors $c = \Omega(\log(n))$ car sinon la probabilité d'obtenir g' est supérieure à $1/|\mathcal{G}_{V+u}| = n^{-k}\frac{1}{|\mathcal{G}_V|}\frac{k!}{e^k}(1+\mathcal{O}(1/n))$ pour n assez grand. Nous venons de montrer que toute exécution qui utilise moins de k appels à `RV`, utilise au moins $\Omega(\log(n))$ bits aléatoires, donc au moins autant de pas de calculs.

Comme nous avons supposé que ces exécutions ont probabilité totale au moins $\varepsilon > 0$, la complexité moyenne de l'algorithme est $\Omega(\log(n))$ ce qui contredit notre hypothèse de départ.

□

5. Cette composante est, avec probabilité proche de 1, le graphe tout entier.

3.4 Propriétés des algorithmes

Nous avons présenté plusieurs algorithmes effectuant la tâche de maintenance pour les graphes k-sortants uniformes. Tous ces algorithmes préservent la distribution uniforme, pourtant ils ne sont pas équivalents : ils diffèrent au minimum de par leur coût respectif, et bien entendu leur implémentation.

Nous présentons dans cette section d'autres caractéristiques qui distinguent ces algorithmes :

— la notion de dualité (présentée §1.1.3) : nous montrons que les algorithmes naturels sont duaux ;

— les changements topologiques entraînés par l'application de l'algorithme.

3.4.1 Dualité des algorithmes

Nous allons prouver que les algorithmes d'insertion et suppression naturels pour les graphes k-sortants uniformes sont effectivement duaux.

Proposition 3.15. SUP-NAT-K-SORTANT *et* INS-NAT-K-SORTANT *sont duaux pour la distribution uniforme des graphes k-sortants.*

Preuve Soit $V \subset \Omega$, $n = |V|$, $u \in \Omega \setminus V$, et deux graphes g et g' tels que $g \in \mathcal{G}_V$ et $g' \in \mathcal{G}_{V+u}$; on note $\mathcal{I}$ l'algorithme d'insertion considéré et $\mathcal{D}$, celui de suppression.

Si g et g' diffèrent plus que sur le voisinage de u dans g' (*i.e.* si $N_g^+(v) \neq N_{g'}^+(v)$ pour un $v \in V \setminus N_{g'}^-(u)$), alors $\mathbb{P}(\mathcal{I}(g,u) = g') = \mathbb{P}(\mathcal{D}(g',u) = g) = 0$ et l'équation de la définition 1.7 est validée.

Ceci est encore le cas si $|N_{g'}^+(v) \Delta N_g^+(v)| \neq 1$ pour $v \in N_{g'}^-(u)$.

Soient maintenant $g \in \mathcal{G}_V$ et $g' \in \mathcal{G}_{V+u}$ ne tombant pas dans les cas précédents. Soit $\ell = |N_{g'}^-(u)|$, nous avons $\mathbb{P}(\mathcal{I}(g,u) = g') = (n-k)^{n-\ell}/n^n \cdot 1/\binom{n}{k}$ d'après la preuve de la proposition 3.8.

De plus, il est facile de voir que $\mathbb{P}(\mathcal{D}(g',u) = g) = 1/(n-k)^\ell$ (cf. preuve de la proposition 3.1).

Comme $\mu_{V+u}(g') = \mu_V(g)((n-k)/n)^n/\binom{n}{k}$, nous obtenons

$$\mu_V(g)\mathbb{P}(\mathcal{I}(g,u) = g') = \mu_{V+u}(g')/(n-k)^\ell = \mu_{V+u}(g')\mathbb{P}(\mathcal{D}(g',u) = g).$$

□

À l'inverse des algorithmes naturels, les algorithmes $\mathcal{I}_2$ =INS-SUCC-K-SORTANT et $\mathcal{I}_3$ =INS-PRED-K-SORTANT ne sont pas duaux avec $\mathcal{D}_1$ =SUP-NAT-K-SORTANT (de même, les deux autres algorithmes de suppression ne le sont pas avec INS-NAT-K-SORTANT).

Un contre exemple simple consiste en un graphe g' possédant un sommet u avec un unique prédécesseur v et tel que $N_g^+(u) \cap N_g[v] = \emptyset$. On considère le graphe g' identique à g privé de u, et où $N_{g'}^+(v)$ diffère de $N_g(v)$ par un seul sommet v', qui non plus n'apparaît pas dans $N_g^+(u)$. On a $\mathbb{P}(\mathcal{I}_2(g',u) = g) = 0$, de même pour $\mathcal{I}_3$, or $\mathbb{P}(\mathcal{D}_1(g,u) = g') = 1/(n-k)$.

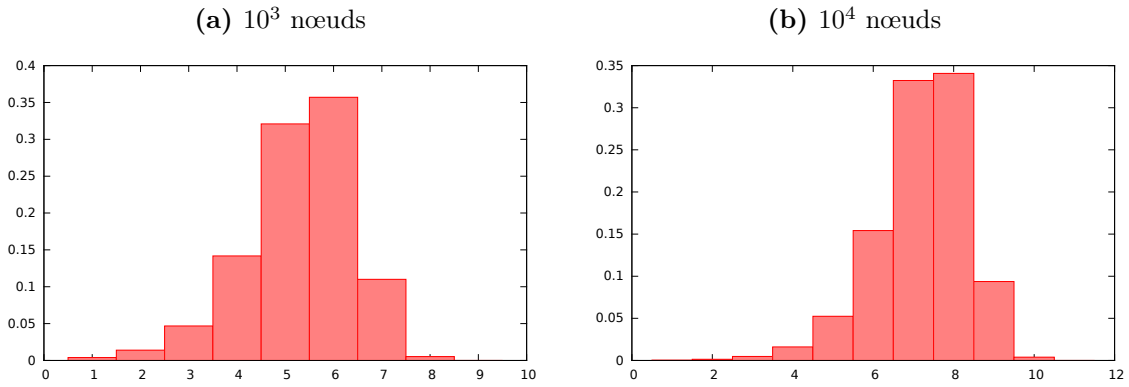


Figure 3.11 – Répartition des distances dans un graphe G_n^2 lors d'une simulation.

3.4.2 Changements topologiques

Les algorithmes de mise à jour que nous avons décrits pour les graphes k-sortants possèdent des coûts, en nombre d'appels moyen à `RV`, différents. En plus de cela, les algorithmes ont un impact significativement différent sur la topologie du graphe après chaque mise à jour.

La propriété de *préservation d'aléa* commune aux différents algorithmes présentés au cours de ce chapitre nous indique que la distribution d'un graphe obtenu après une suppression (resp. insertion) est identique pour les trois algorithmes de suppression (resp. d'insertion). Néanmoins, le graphe obtenu après une mise à jour est *dépendant* du graphe avant mise à jour (peu d'arêtes sont en effet modifiées au cours de l'opération). Ainsi, l'utilisation d'un algorithme de suppression ou d'insertion particulier, bien qu'ils préservent tous la distribution souhaitée, aura un impact distinct des autres *localement* autour du sommet supprimé/inséré. Toutefois, cet impact local entraîne aussi des modifications globales, par exemple sur l'évolution des distances entre sommets.

Les changements de distances se révèlent pertinent à considérer dans le cas de réseaux décentralisés, car ils peuvent avoir différentes conséquences négatives au niveau du réseau (routage, partage de ressources, etc).

À titre indicatif, la répartition des distances dans un graphe G_n^2, lors d'une simulation, est donné figure 3.11 pour les deux tailles (10^3 et 10^4 nœuds) de graphe analysées dans cette section. Cette simulation indique notamment que peu de paires de nœuds sont distants du diamètre dans le graphe, qui est la plupart du temps de 9 pour $n = 10^3$ et 11 pour $n = 10^4$ (voir table 3.2).

Afin de mesurer l'impact des mises à jour, nous proposons de comparer trois statistiques liées à l'évolution des distances non orientées entre un graphe G (graphe non-orienté sous-jacent au graphe k-sortant) et un graphe G', obtenu après une mise à jour :

— le nombre de distances modifiées, *i.e.* le nombre de paires $\{u,v\}$ tel que $d_G(u,v) \neq d_{G'}(u,v)$;

— la somme des modifications de distances, *i.e.* $\frac{1}{2}\sum_{u,v\in V^2}|d_G(u,v)-d_{G'}(u,v)|$;
— le nombre de distances modifiées par plus d'une unité, *i.e.* le nombre de paires $\{u,v\}$ tel que $|d_G(u,v)-d_{G'}(u,v)|\geq 2$.

Méthodologie des simulations

Les graphiques présentés ont été obtenu en analysant au moins 10^4 simulations de graphes k-sortants indépendants. Chaque simulation consiste en la génération d'un graphe k-sortant uniforme, puis la suppression d'un sommet du graphe [6] ou l'insertion d'un nouveau sommet.

Pour chaque paire $\{u,v\}$ de sommets de $V(G)\cap V(G')$ possible, les distances $d_G(u,v)$ et $d_{G'}(u,v)$ sont calculées. Les trois différentes statistiques sont alors collectées en sommant sur toutes les paires possibles. Chaque simulation fait apparaître le degré entrant du sommet supprimé/ajouté. Les simulations concernant les insertions ont été réalisées séparément de celles concernant les suppressions.

Nous présentons ici deux figures résumant les simulations :

— **Figure 3.12** : distribution du nombre de distances modifiées après une suppression (fonction escalier), classées selon le degré entrant (histogrammes) lors de 10^5 simulations de graphes G_n^2 indépendants pour $n=10^3$ (colonne de gauche) et $2{,}5\cdot 10^4$ simulations de graphes G_n^2 indépendants pour $n=10^4$ (colonne de droite) ; la taille des classes est de 50 (10^3 sommets) et 750 (10^4 sommets) ; les courbes de la dernière ligne représentent les fonctions de densité cumulative pour les trois algorithmes.

— **Figure 3.13** : distribution du nombre de distances modifiées par plus d'une unité après une mise à jour sous forme d'histogramme (figures de gauche) et de fonction de densité cumulative (figures de droite) lors de 10^4 simulations de graphes G_n^2 indépendants pour $n=10^3$ et $n=10^4$; la taille des classes des histogrammes est 10 pour 10^3 nœuds et 75 pour 10^4 nœuds.

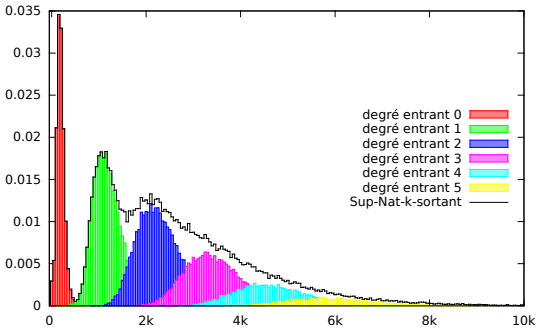
Analyse

Lors de la suppression d'un nœud dans un graphe k-sortant, un paramètre important contrôlant la modification relative du graphe est le degré entrant du sommet supprimé : il va déterminer la quantité de sommets impactés par la mise à jour.

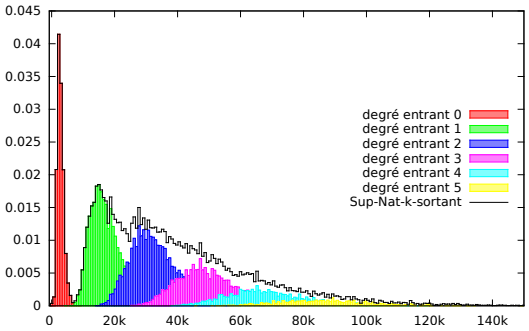
La table 3.3 résume la moyenne et l'écart-type du nombre de distances modifiées pour trois degrés entrants (les données sont celles présentées par la figure 3.12). Les valeurs successives sont celles obtenues après une suppression par l'algorithme SUP-NAT-K-SORTANT, SUP-SUCC-K-SORTANT et SUP-PRED-K-SORTANT ; les valeurs sont tronquées à l'entier inférieur.

6. Le sommet supprimé est le dernier ajouté, mais ça n'a pas d'importance sachant la propriété d'invariance par renommage de la distribution considérée.

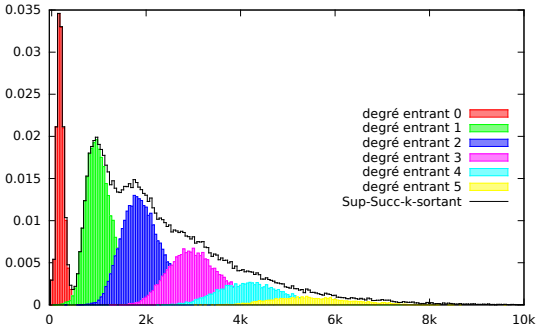
(a) Sup-Nat-k-sortant 10^3 nœuds



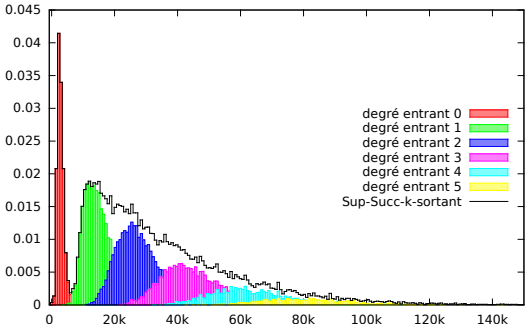
(b) Sup-Nat-k-sortant 10^4 nœuds



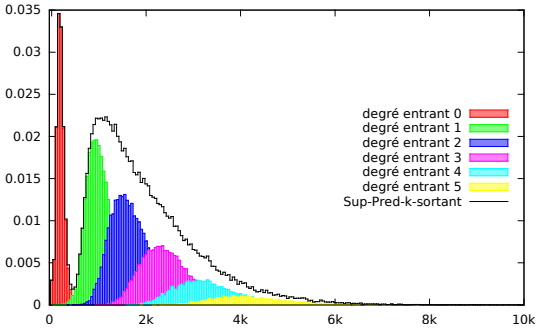
(c) Sup-Succ-k-sortant 10^3 nœuds



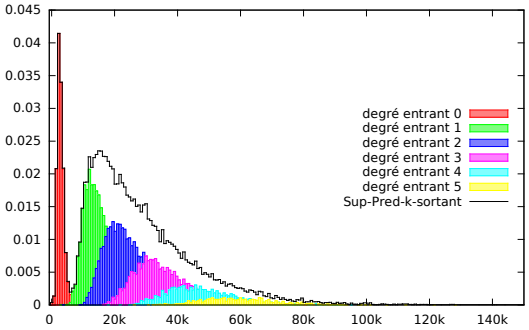
(d) Sup-Succ-k-sortant 10^4 nœuds



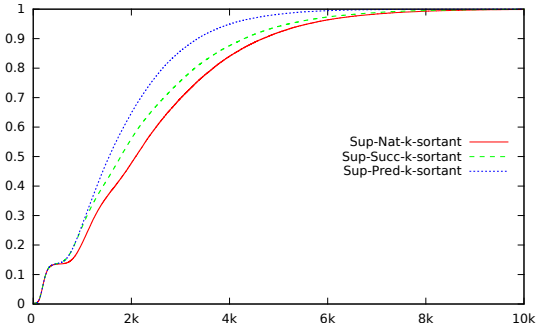
(e) Sup-Pred-k-sortant 10^3 nœuds



(f) Sup-Pred-k-sortant 10^4 nœuds



(g) Cdf 10^3 nœuds



(h) Cdf 10^4 nœuds

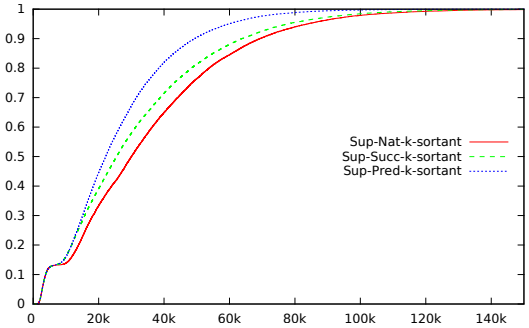
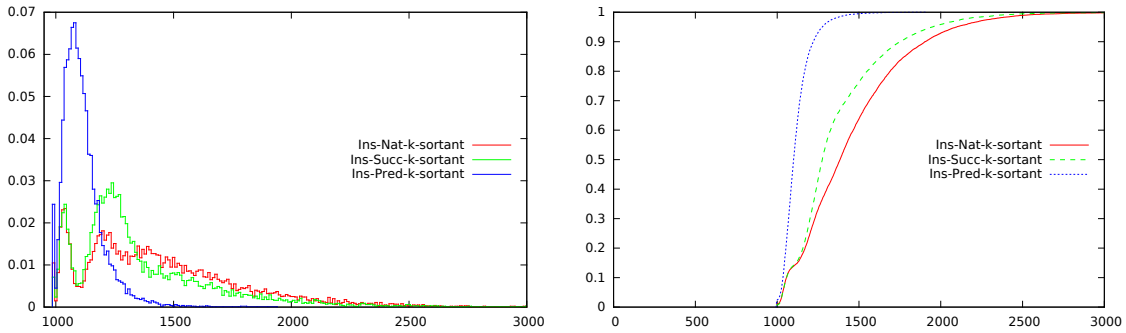
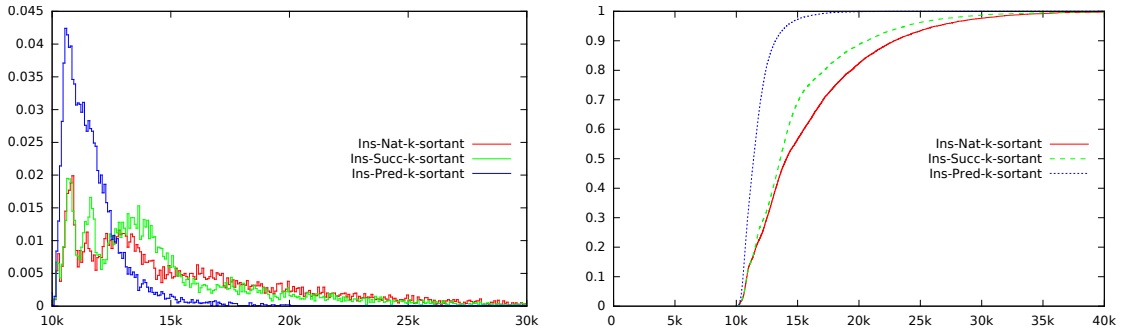


Figure 3.12 – Nombre de distances modifiées après une suppression.

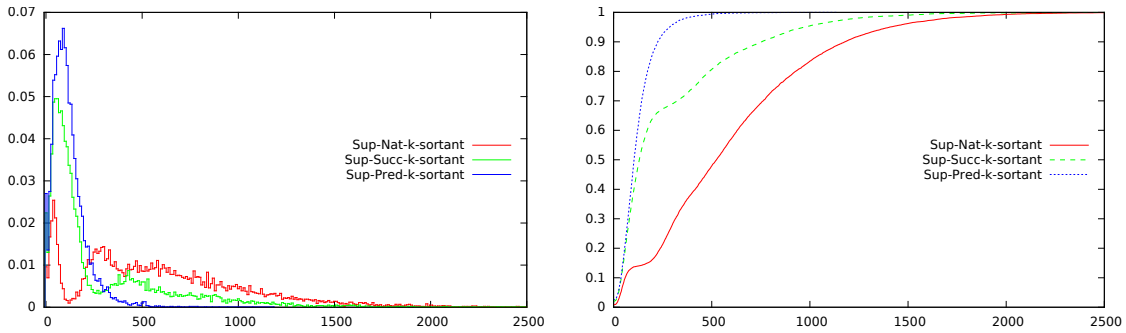
(a) Insertion 10^3 nœuds



(b) Insertion 10^4 nœuds



(c) Suppression 10^3 nœuds



(d) Suppression 10^4 nœuds

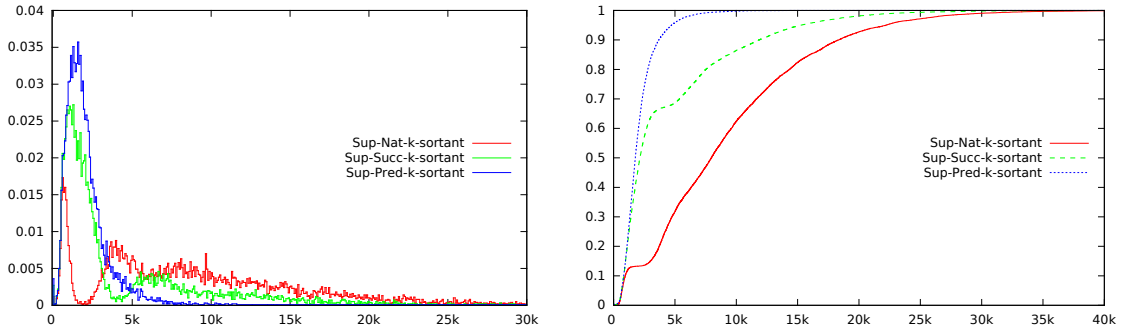


FIGURE 3.13 – Nombre de distances modifiées par plus d'une unité.

	Moyenne		Écart-type	
δ^-	10^3	10^4	10^3	10^4
0	210	3201	81	1059
1	1220; 1064; 1061	17545; 15227; 15223	920; 812; 808	5062; 4749; 4733
2	2299; 1965; 1709	33242; 28235; 24391	2353; 2020; 1770	7865; 7311; 7259
3	3461; 3101; 2474	50292; 45007; 35498	4271; 3833; 3072	10202; 9792; 8942
tous	2373; 2120; 1765	34860; 31120; 25650	1705; 1596; 1193	25633; 24072; 17856

Table 3.3 – Statistiques sur le nombre de distances modifiées.

Les simulations effectuées (cf. figure 3.12, table 3.3) nous indiquent que lorsque l'on considère le nombre de distances modifiés, l'algorithme de suppression Sup-Pred-k-sortant a un impact significativement moindre que les deux autres algorithmes de suppression présentés (en moyenne environ 25% de distances en moins modifiées par rapport à la suppression naturelle, environ 17% par rapport au second algorithme).

En outre comme le montre les simulations, pour chaque degré entrant, l'impact de l'algorithme sera relativement moins important (moyenne et écart-type plus faible pour chaque degré entrant observé). Enfin, lorsque le degré entrant du sommet supprimé est 0, les trois algorithmes se comportent identiquement.

Ce phénomène apparaît clairement pour plusieurs tailles de graphe, pour différentes valeurs de k, et de manière similaire lors de l'insertion ; voir § A.2 en annexe pour des jeux de tests complémentaires.

Les distributions obtenues en considérant la somme des modifications de distances (en valeur absolue) présentent la même forme et les mêmes caractéristiques qu'en considérant le nombre de distances modifiés (voir figure A.21, en annexe). Ceci s'explique par le fait que les distances sont rarement modifiées par plus d'une unité.

Si nous nous intéressons de plus près seulement aux impacts importants (distances modifiées par plus d'une unité), les différences de conséquence entre les algorithmes sur le graphe sont encore plus marquées.

D'après la figure 3.13 et table 3.4, le troisième algorithme d'insertion (et celui de suppression) possèdent encore des conséquences sur les changement de distances largement moindre que les deux autres en moyenne : environ $22-26\%$ de distances en moins modifiées par rapport à l'insertion naturelle et environ $77-80\%$ de distances en moins modifiées par rapport à la suppression naturelle.

De plus, le deuxième algorithme de suppression, Sup-Succ-k-sortant, présente un impact global plus intermédiaire (environ 51% de distances modifiées en moins par rapport à la suppression naturelle) que dans le cas de l'insertion (où il possède

	Moyenne		Écart-type	
Algorithmes	$\mathbf{10^3}$	$\mathbf{10^4}$	$\mathbf{10^3}$	$\mathbf{10^4}$
INS-NAT-K-SORTANT	1448	15813	339	5195
INS-SUCC-K-SORTANT	1361	14796	290	4423
INS-PRED-K-SORTANT	1118	11712	85	1280
SUP-NAT-K-SORTANT	594	9186	433	6734
SUP-SUCC-K-SORTANT	280	4502	328	5104
SUP-PRED-K-SORTANT	122	2087	88	1366

TABLE 3.4 – Statistiques sur le nombre de distances modifiées par plus d'une unité.

un impact comparable à l'insertion naturelle, avec environ 6% de différence).

Remarquons enfin que sur les deux tailles de graphe testées, le nombre moyen de changement de distances est toujours compris entre $1{,}5n$ et $3{,}5n$ pour les différents algorithmes (cf. table 3.3), ce qui représente une proportion faible des $\mathcal{O}(n^2)$ distances du graphe.

Pour conclure, l'observation de telles différences de comportement entre algorithmes encourage le développement futur d'une étude théorique approfondie sur les modifications topologiques de différents algorithmes de mise à jour pour une même distribution cible de graphes aléatoires.

3.5 Graphes μ-sortants

Nous avons présenté jusqu'à présent des algorithmes pour la tâche de maintenance sur les graphes k-sortants uniformes, possédant des propriétés différentes, et dont les meilleurs sont optimaux concernant leur coût. Nous proposons dans cette section de généraliser le modèle en permettant au degré sortant d'un sommet de varier selon une distribution de probabilité μ à support fini, connue et fixé à l'avance.

Les graphes μ-sortants sont une extension des graphes k-sortants uniformes : la taille de chaque voisinage sortant, à la place d'être homogène, est choisie selon la distribution μ et indépendamment pour chaque sommet ; chaque voisinage est ensuite uniformément et indépendamment choisi sur V privé du sommet source.

Définition 3.2. Soit μ une distribution de probabilité à support fini sur les entiers. Le graphe $G = (V,E)$ suit la distribution s_V^μ des graphes μ-sortants sur V si :

— les $(N_G^+(v))_{v\in V}$ sont des variables aléatoires indépendantes, et

— pour tout $v \in V$, $\mathbb{P}(|N_G^+(v)| = k) = \mu(k)$, et

— conditionnellement à $|N_G^+(v)| = k$, l'ensemble $N_G^+(v)$ est uniforme sur les k-sous-ensembles de $V - v$.

Notation On dénote $|\mu|$ l'élément maximum du support de μ (qui rappelons-le doit être fini), *i.e.* $|\mu| = \max\{i \in \mathbb{N} \mid \mu(i) > 0\}$. La famille de distributions est, quant à elle, notée $s^\mu = (s_V^\mu)_{V \subset \Omega}$.

Nous travaillons ici avec le même ensemble possible de graphes $\mathcal{G}_V$ que précédemment, les graphes orientés sans boucles. L'ensemble $\mathcal{G}_V^\mu \subset \mathcal{G}_V$ représente les graphes *atteignables*, *i.e.* $\mathcal{G}_V^\mu = \mathrm{Supp}(s_V^\mu)$, qui sont exactement les graphes orientés dont les degrés sortants des sommets appartiennent au support de μ. Afin d'éviter que la construction ne soit impossible, on pose $\mathcal{G}_V^\mu = \emptyset$ si $|V| \leq |\mu|$.

Remarque Si $G = (V,E)$ suit la loi s_V^μ, alors pour tout $N \subset V$ et $v, v' \in V$

$$\mathbb{P}(N^+(v) = N) = \begin{cases} 0 & \text{si } v \in N \\ \dfrac{\mu_{|N|}}{\binom{n-1}{|N|}} & \text{sinon} \end{cases} \quad \text{et} \quad \mathbb{P}((v,v') \in E) = \frac{\mathbb{E}(\mu)}{n-1},$$

et le degré entrant d'un sommet est distribué selon la loi $\mathrm{Bin}(n-1, \mathbb{E}(\mu)/(n-1))$ car les évènements $u \in N_G^+(v)$ pour $v \in V - u$ sont indépendants. La probabilité d'obtenir un graphe $g \in \mathcal{G}_V^\mu$ en particulier est

$$\prod_{v \in V} \frac{\mu(|N_g^+(v)|)}{\binom{n-1}{|N_g^+(v)|}}.$$

3.5.1 Algorithmes de suppression

Les algorithmes présentés section 3.2 (légèrement modifié pour le dernier) préservent la distribution des graphes μ-sortants. Nous avons pris le parti ici de conserver lors d'une suppression les degrés entrants des sommets non supprimés. Cette conservation nous permet lors d'une suppression de ne modifier (ajout/suppression) qu'un nombre minimal d'arêtes, comme ce fut le cas pour les graphes k-sortants.

Proposition 3.16. SUP-NAT-K-SORTANT *préserve* s^μ.

Preuve Soit $G = (V, E) \sim s_V^\mu$ avec $n = |V|$ et $u \in V$ les entrées de l'algorithme. Dans le but de montrer $G' \sim s_{V'}^\mu$, nous prouvons dans un premier temps que les $(N_{G'}^+(v))_{v \in V'}$ sont mutuellement indépendants, et que pour tout v, $\mathbb{P}(N_{G'}^+(v) = N) = \mu(|N|)/\binom{n-2}{|N|}$ pour $N \subset V - u - v$.

Nous reprenons les définitions de $R_{i,v}$ et I_v données dans la preuve de la proposition 3.3. Par définition de `RVA`, on a $X \sim R_{I_v,v}$ à la ligne 4 au tour de boucle de v.

En analysant l'algorithme, nous remarquons la relation suivante pour $v \in V - u$,

$$N^+_{G'}(v) = \begin{cases} N^+_G(v) & \text{si } u \notin N^+_G(v) \\ N^+_G(v) - u + R_{I_v,v} & \text{sinon} \end{cases}$$

Chaque voisinage $N^+_{G'}(v)$ est donc une fonction déterministe de la paire $P_v = \langle N^+_G(v), (R_{i,v})_{i\geq 1}\rangle$. Puisque les paires P_v, pour $v \in V - u$, sont indépendantes, les ensembles $(N^+_{G'}(v))_{v\in V-u}$ aussi.

Soient $v \in V - u, N \subset V - u - v$ et $k = |N|$.

Remarquons que $\mathbb{P}(N^+_{G'}(v) = N, u \notin N^+_G(v)) = \mathbb{P}(N^+_G(v) = N)$, et

$$\mathbb{P}(N^+_{G'}(v) = N, u \in N^+_G(v)) = \sum_{w\in N} \mathbb{P}(N^+_G(v) = N - w + u, R_v = w)$$

d'où $\mathbb{P}(N^+_{G'}(v) = N, u \in N^+(v)) = k \cdot \frac{\mu(k)}{\binom{n-1}{k}} \cdot \frac{1}{n-1-k} = \frac{\mu(k)}{\binom{n-1}{k}} \cdot \frac{k}{n-1-k}$.

Au total, $\mathbb{P}(N^+_{G'}(v) = N) = \mu(k)(1 + \frac{k}{n-1-k})/\binom{n-1}{k} = \mu(k)/\binom{n-2}{k}$. □

Proposition 3.17. *Le coût asymptotique sur $|V|$ de* SUP-NAT-K-SORTANT *est $\mathbb{E}(\mu)$ sur les graphes μ-sortants.*

Preuve $\mathbb{E}(\mu)$ est la moyenne du nombre de prédécesseurs, donc le coût moyen est $\mathbb{E}(\mu)(1 + \mathcal{O}(1/|V|))$ en prenant compte des rejets possibles. □

Le deuxième algorithme SUP-SUCC-K-SORTANT proposé pour les graphes k-sortants uniformes préserve lui aussi la distribution des graphes μ-sortants.

La preuve de sa correction est une combinaison des preuves des propositions 3.3 et 3.16. L'indépendance des $(N^+(v))_{v\in V'}$ est donnée par la première partie de la preuve de préservation pour SUP-SUCC-K-SORTANT, et l'uniformité est obtenue en remarquant que $R_v \sim R_{I_v,v}$ et en refaisant les calculs de la preuve de la proposition 3.16.

Nous pouvons aussi proposer un troisième algorithme de suppression, qui est quasiment identique à SUP-PRED-K-SORTANT mais doit gérer le cas où $\delta^+(u) = 0$ lors du départ de u (L-ème tour de boucle). Une solution consiste à utiliser directement `RV` dans ce cas précis, ce qui aboutira à un algorithme dont le coût est au minimum de $\mu(0)\mathbb{P}(\delta^-(u) > 0)$.

Une astuce pour contourner ce problème est de remplacer le voisinage du premier prédécesseur v de u par celui de u, puis utiliser une fois `RV` pour compléter ce voisinage s'il contient v.

L'algorithme proposé reprend SUP-PRED-K-SORTANT mais remplace le cas où $i = L$ par le pseudo-code suivant :

si $i = L$ **alors**
  $N \leftarrow N^+_G(u)$
  si $u_L \in N$ **alors**
    $N \leftarrow N - u_L +$ **RandomVertexAvoiding**(N)
  $E' \leftarrow E' \setminus \{(u_L, v) \in E\} \cup \{(u_L, v) \mid v \in N\}$

3.5.2 Algorithmes d'insertion

Une simple adaptation de l'algorithme INS-NAT-k-SORTANT permet de maintenir les graphes μ-sortants. Le voisinage sortant du nouveau sommet u doit d'abord être construit en tirant sa taille K selon μ, puis en choisissant un K-sous-ensemble de V uniforme via `RandomSubset()` par exemple. Contrairement aux graphes k-sortants, $\mathrm{Bin}(|V|, \mathbb{E}(\mu)/|V|)$ n'est pas la bonne distribution pour choisir le degré entrant de u si l'on souhaite conserver les degrés sortants des sommets déjà présents : cette dernière doit dépendre de la taille des voisinages courants du graphe.

L'astuce utilisée est alors de tirer une binomiale avec une probabilité de succès plus forte $\mathrm{Bin}(|V|, |\mu|/|V|)$ où $|\mu|$ est le degré sortant maximum possible (cf. notation page 101 après la définition 3.2), puis de ne conserver chaque sommet sélectionné v qu'avec probabilité $\delta^+(v)/|\mu|$. La preuve de correction démontre la dualité d'un tel algorithme avec SUP-NAT-K-SORTANT appliqué aux graphes μ-sortants.

Algorithme 17 INS-NAT-μ-SORTANT

Entrées: un graphe μ-sortant $G = (V, E)$, et un sommet $u \notin V$
Sortie: un graphe μ-sortant G'

$V' \leftarrow V + u$
$K \leftarrow \mu()$
$S \leftarrow \textbf{RandomSubset}(K)$
$E' \leftarrow E \cup \{(u, v) \mid v \in S\}$
$L \leftarrow \mathit{Binomiale}(|V|, |\mu|/|V|)$
$W \leftarrow \textbf{RandomSubset}(L)$
pour tout $v \in W$ **faire**
    si $\mathit{Bernoulli}(|N_G^+(v)|/|\mu|)$ **alors**
        $X \leftarrow \mathit{Uniforme}(N_G^+(v))$
        $E' \leftarrow E' - (v, X) + (v, u)$
$G' \leftarrow (V', E')$

Proposition 3.18. INS-NAT-μ-SORTANT *préserve* s^μ.

Preuve Nous prouvons que l'algorithme d'insertion $\mathcal{I}$ présenté et $\mathcal{D}$ =SUP-NAT-K-SORTANT sont duaux sur les graphes μ-sortants, ce qui permet de conclure via les propositions 3.16 et 1.4.

Soient $V \subset \Omega$, $n = |V|$, $u \in \Omega \setminus V$, $g \in \mathcal{G}_V$ et $g' \in \mathcal{G}_{V+u}$, choisis de telle sorte que $|N_{g'}^-(v) \Delta N_g^-(v)| = 1$ pour $v \in N_{g'}^-(u)$ et tous les autres voisinages sont identiques dans g et g'. Remarquons

$$\frac{s^\mu_{V+u}(g')}{s^\mu_V(g)} = \frac{\mu(\delta^+(u))}{\binom{n}{\delta^+(u)}} \prod_{v \in V} \frac{n - \delta^+(v)}{n}.$$

De plus, il est aisé de calculer $\mathbb{P}(\mathcal{D}(g', u) = g) = \prod_{v \in N_{g'}^-(u)} 1/(n - \delta^+(v))$.

Il ne reste donc plus qu'à obtenir $\mathbb{P}(\mathcal{I}(g,u) = g')$ (il est évident que pour les autres couples de graphes possibles, chaque algorithme a une probabilité nulle de passer de l'un à l'autre).

Pour chaque v, soit B_v une variable de Bernoulli indiquant si $u \in N^+_{G'}(v)$ à la fin de l'algorithme INS-NAT-μ-SORTANT, donc $B_v = 1$ avec probabilité $|\mu|/n \cdot \delta^+(v)/|\mu| = \delta^+(v)/n$. En considérant la binomiale ligne 5 comme une somme de Bernoulli B'_v indépendantes, les B_v sont indépendants en tant que fonction déterministe de B'_v et des bits indépendants utilisés par la `Bernoulli` ligne 8.

Ce qui nous permet de calculer $\mathbb{P}(N^-_{G'}(u) = N)$ pour $N \subset V$,

$$\mathbb{P}(N^-_{G'}(u) = N) = \prod_{v \in N} \frac{\delta^+(v)}{n} \prod_{v \in V \setminus N} \frac{n - \delta^+(v)}{n},$$

ainsi que la probabilité conditionnelle suivante pour $N_v \subset V - v$ et $x \in N_v$

$$\mathbb{P}(N^-_{G'}(v) = N_v - x + u, \forall v \in N \mid N^-_{G'}(u) = N, N^+_G(v) = N_v) = \prod_{v \in N} \frac{1}{\delta^+(v)}.$$

Nous pouvons à présent conclure par

$$\mathbb{P}(\mathcal{I}(g,u) = g') = \frac{\mu(\delta^+(u))}{\binom{n}{\delta^+(u)}} \frac{1}{n^n} \prod_{v \in V \setminus N^-_{g'}(u)} n - \delta^+(v) = \frac{s^\mu_{V+u}(g')}{s^\mu_V(g)} \mathbb{P}(\mathcal{D}(g',u) = g).$$

□

Proposition 3.19. *Le coût moyen de* INS-NAT-μ-SORTANT *est asymptotiquement de* $|\mu| + \mathbb{E}(\mu)$ *(lorsque* $|V|$ *tend vers l'infini).*

Preuve Le premier `RandomSubset()` utilise en moyenne $\mathbb{E}(\mu)(1 + \mathcal{O}(1/n))$ appels à la primitive `RV`, quant au second, il effectue $|\mu|(1 + \mathcal{O}(1/n))$ appels en moyenne à la primitive.

□

Si on suit le même type d'adaptation pour les autres algorithmes d'insertion, nous obtenons un coût moyen d'insertion, en nombre d'appels à `RV`, supérieur à $|\mu|$, ce qui n'est pas très satisfaisant lorsque $|\mu|$ est très largement supérieur à $\mathbb{E}(\mu)$. Un problème intéressant restant ouvert est d'obtenir un algorithme d'insertion avec un coût de décentralisation proche de $\mathbb{E}(\mu)$.

3.5.3 Degrés non bornés

Nous terminons ce chapitre en évoquant succinctement le cas où la distribution des degrés n'est pas bornée *a priori*, généralisant le modèle de graphes μ-sortants. Nous surchargeons la notation et désignons ces graphes par ρ-sortants. L'idée est que la loi ρ est utilisée pour tirer un « degré sortant idéal » pour chaque sommet, le degré réel étant le plus petit parmi $|V| - 1$ et le degré idéal.

Définition 3.3. Soit ρ une distribution de probabilité sur les entiers.
Nous avons $G = (V,E) \sim s_\rho$ si :
— les $(N_G^+(v))_{v\in V}$ sont des variables aléatoires indépendantes, et
— pour tout $v \in V$, $\mathbb{P}(|N_G^+(v)| = k) = \rho(k)$ pour $k < n-1$ et $\mathbb{P}(|N_G^+(v)| = |V|-1) = 1 - \sum_{k=0}^{|V|-2} \rho(k)$, et
— conditionnellement à $|N_G^+(v)| = k$, $N_G^+(v)$ est uniforme sur les k-sous-ensembles de $V - v$.

Pour cette distribution de graphes aléatoires, l'algorithme Sup-Nat-k-sortant est aussi un algorithme de suppression préservant la loi. Pour le montrer, nous pouvons reprendre la preuve de la proposition 3.16 pour un graphe en entrée de taille n. L'indépendance des nouveaux voisinages est obtenue de manière analogue et le calcul de la probabilité d'obtenir un voisinage particulier est lui aussi identique pour toute taille jusqu'à $n-3$. Celle d'obtenir un voisinage de taille $n-2$ est exactement

$$\rho(n-2) + \mathbb{P}(\delta_G^+(v) = n-1) = 1 - \sum_{k=0}^{n-3} \rho(k),$$

car il n'existe qu'un unique voisinage possible de taille $n-2$ lors de la suppression d'un voisin d'un sommet de degré $n-1$. Le coût d'une telle suppression est le nombre moyen de prédécesseurs du sommet supprimé, soit asymptotiquement $\mathbb{E}(\rho)$.

Considérons maintenant l'insertion d'un nouveau sommet. Si on adapte l'algorithme Ins-Nat-μ-sortant en remplaçant la loi μ par la loi ρ, nous obtenons une procédure qui consiste à passer tous les sommets en revue. Cet algorithme n'est réalisable qu'en énumérant les sommets, une tâche elle-même qui n'est envisageable qu'avec la connaissance de $|V|$. La question de la maintenabilité de cette distribution sans la connaissance de n reste ouverte, ainsi que la possibilité, même pour des cas particuliers de loi ρ, de maintenir exactement les graphes ρ-sortants au moyen d'algorithmes efficaces.

À l'heure actuelle, nous pouvons résumer nos connaissances sur la maintenance de graphes ρ-sortant ainsi :
— il existe un algorithme de suppression préservant la distribution s_ρ nécessitant asymptotiquement $\mathbb{E}(\rho)$ appels à `RV` en moyenne ;
— il existe un algorithme d'insertion utilisant $|V|$ préservant la distribution s_ρ nécessitant $\mathcal{O}(|V|\log(|V|))$ appels à `RV` en moyenne (en énumérant les sommets) ;
— aucun algorithme n'est connu lorsque $|V|$ n'est pas accessible.

Chapitre 4

Simulation en aveugle de lois binomiales

Sommaire

Dans le chapitre précédent, il a été montré comment maintenir efficacement la distribution des graphes k-sortants uniformes (et μ-sortants) en utilisant explicitement la taille courante du graphe à mettre à jour. Le plus simple de ces algorithmes de suppression n'a pas besoin d'utiliser la taille du graphe et opère en utilisant relativement peu d'appels à la primitive de décentralisation `RV`, soit k appels en moyenne asymptotiquement pour les graphes k-sortants uniformes et $\mathbb{E}(\mu)$ pour les μ-sortants. En ce qui concerne l'algorithme d'insertion le plus simple présenté, il utilise explicitement la taille lors de la simulation de la binomiale Bin$(|V|, k/|V|)$, ou Bin$(|V|, |\mu|/|V|)$ pour les μ-sortants. Ce recours à la simulation de lois binomiales est également présent dans les algorithmes d'insertion plus complexes, et il n'est pas clair qu'il soit possible de s'en passer.

Ce chapitre explore comment ces distributions binomiales peut être simulée en utilisant uniquement des appels à `RV`, et ce sans avoir accès à $|V|$, la taille du graphe avant insertion. De plus, cette simulation n'entraînera aucun surcoût pour l'algorithme INS-NAT-K-SORTANT (et INS-SUCC-K-SORTANT), car un ensemble de sommets uniforme de taille binomiale sera aussi obtenu par la même occasion (économisant l'appel à `RandomSubset` pour générer l'ensemble des prédécesseurs).

Ce chapitre est divisé en quatre parties ayant pour toile de fond la simulation de la loi binomiale $\mathrm{Bin}(n, k/n)$ à partir de la primitive `RV` (générateur sur un ensemble V de taille n) sans connaître n (et *a fortiori* V). Le cœur du chapitre est basé sur une nouvelle construction combinatoire, qui a son intérêt propre au delà de son application à notre problème de simulation.

La première partie montre comment simuler la loi $\mathrm{Bin}(n, k/n)$ à partir uniquement d'un générateur de lois uniformes sur $[\![n]\!]$. Le but est de montrer que cette loi est simulable dans un modèle plus fort (chaque tirage nous donne directement une borne inférieure sur n) et ce de manière très efficace : en utilisant un unique tirage pour $\mathrm{Bin}(n, 1/n)$ et un tirage en moyenne asymptotiquement pour $k \geq 2$ (en ayant accès à une source indépendante de bits aléatoires). Elle permet de montrer que la simulation est possible dans notre modèle plus faible de départ (car les uniformes sur $[\![n]\!]$ sont elles-même simulables), et d'introduire plusieurs notions et intuitions qui seront utilisées dans le reste du chapitre, ainsi que dans le chapitre suivant.

La seconde section présente un algorithme de simulation pour la distribution $\mathrm{Bin}(n, 1/n)$ à partir de tirages d'uniformes sur un ensemble V avec $n = |V|$. L'algorithme présenté nécessite de simuler le nombre de points fixes d'une permutation aléatoire uniforme. Cette partie est résolue ici en générant une permutation complète de la taille souhaitée, puis en renvoyant son nombre de points fixes.

La troisième section présente comment générer le nombre de points fixes d'une permutation de taille n en au plus 3,2 tirages d'uniformes en moyenne sur un ensemble de taille n ; cette méthode permet ainsi de résoudre efficacement la simulation effectuée à la section précédente. En outre, cette troisième partie, essentiellement combinatoire, introduit un nouvel arbre de génération pour les permutations ayant plusieurs propriétés intéressantes. En particulier, le nombre de points fixes est *préservé* (au maximum) lors d'une descente aléatoire uniforme dans l'arbre. Dans cette section, nous décrirons précisément la construction de cet arbre de génération, ses propriétés, ainsi que plusieurs applications potentielles qui ne concernent pas la simulation de la loi binomiale dans notre contexte distribué.

Enfin, la dernière section fait le lien entre l'arbre de génération décrit et notre problème de simulation. En exploitant astucieusement certaines propriétés de l'arbre présenté précédemment, nous montrons dans cette partie qu'il est possible de simuler efficacement la loi binomiale paramétrée par n et k/n.

4.1 Simulation optimale à partir d'uniformes

Cette section, servant d'introduction au problème de simulation posé, est consacrée à montrer comment simuler la loi $\mathrm{Bin}(n, k/n)$ à partir d'uniformes sur $[\![n]\!]$, sans connaître n (mais avec la garantie que l'on ait $n \geq k$). Nous supposerons donc la possibilité de générer un entier suivant la distribution $\mathrm{Unif}([\![n]\!])$ (via un appel à une fonction en boîte noire de type `RV`), et expliciterons des algorithmes probabilistes efficaces simulant $\mathrm{Bin}(n, k/n)$. La section se conclut en montrant que la loi $\mathrm{Unif}([\![n]\!])$ peut elle-même être simulée à partir d'un générateur sur un ensemble inconnu de taille n, et ce sans connaître n.

Nous présenterons dans un premier temps la simulation dans le cas $k = 1$, qui n'utilise qu'un seul tirage d'uniforme, avant de présenter le cas $k \geq 2$ qui lui a besoin de $1 + \mathcal{O}(1/n)$ tirages en moyenne.

L'intérêt de cette section est triple :

1. Les algorithmes présentés ici prouvent par des preuves simples la possibilité de simuler la loi $\mathrm{Bin}(n, k/n)$ pour $k \geq 1$ dans notre modèle décentralisé.
2. Les algorithmes résolvent de manière asymptotiquement optimale le problème de simulation, dans un modèle plus simple, mais pertinent, où il existe une bijection connue entre les sommets renvoyés par `RV` et $[\![n]\!]$.
3. Enfin, leur coût permet d'avoir un comparatif pour les algorithmes plus complexes, présentés section 4.4.

Notation Nous notons $b_j^{k,n}$ la probabilité qu'une variable aléatoire distribuée selon la loi binomiale de paramètre n et k/n soit égale à j lorsque $n \geq k \geq 1$, *i.e.*

$$b_j^{k,n} = \binom{n}{j} \left(\frac{k}{n}\right)^j \left(1 - \frac{k}{n}\right)^{n-j} \quad \text{pour} \quad 0 \leq j \leq n$$

et $b_j^{k,n} = 0$ pour $j > n$. Nous dénoterons le cas spécial $b_j^{1,n}$ par $b_j^n = \binom{n}{j} \frac{(n-1)^{n-j}}{n^n}$.

4.1.1 Simulation de la loi $\mathbf{Bin}(n, 1/n)$

Nous proposons la procédure suivante afin de simuler la distribution $\mathrm{Bin}(n, 1/n)$:

Algorithme 18 Bin1

1. Générer m comme une uniforme sur $[\![n]\!]$.
2. Renvoyer $0 \leq j \leq m$ avec probabilité $p_j^m = m \cdot b_j^m - (m-1) \cdot b_j^{m-1}$.

Le lemme suivant permet de garantir que $p_j^m \geq 0$.

Lemme 4.1. *Pour $n \geq 1$ et $j \in [\![0, n]\!]$, nous avons*

$$n \cdot b_j^n \geq (n-1) \cdot b_j^{n-1}.$$

Preuve Les cas $n = 1$ et $j = n$ sont tout deux évidents donc à présent nous ferons l'hypothèse $n \geq 2$ et $j < n$.

Prouvons dans un premier temps le cas particulier $j = 0$.

Nous devons montrer que $n \cdot b_0^n \geq (n-1) \cdot b_0^{n-1}$, *i.e.*

$$\frac{n}{n-1}\left(1-\frac{1}{n}\right)^n \geq \left(1-\frac{1}{n-1}\right)^{n-1}.$$

L'inégalité est satisfaite car la fonction $f : x \mapsto (1-1/x)^x$, qui tend vers $1/e$, est strictement croissante pour $x \geq 2$; une dérivation suffit à le vérifier :

$$f'(x) = \left(1-\frac{1}{x}\right)^x \left(\log\left(\frac{x-1}{x}\right) - \frac{1}{x-1}\right)$$

où la positivité de l'expression découle de $\log(x) - \log(x-1) < \frac{1}{x-1}$.

Le reste de la preuve est une simple induction sur j avec $j = 1$ comme cas de base.

Base : Notons que le cas $n = 2$ est vérifié car $2 \cdot b_1^2 \geq b_1^1 \Leftrightarrow 1 \geq 1$.

Pour $n \geq 3$, nous réécrivons $n \cdot b_1^n \geq (n-1) \cdot b_1^{n-1}$ comme $\frac{n^2(n-2)}{(n-1)^3}\left(\frac{n-1}{n}\right)^n \geq \left(\frac{n-2}{n-1}\right)^{n-1}$. L'inégalité est satisfaite car $\frac{n^2(n-2)}{(n-1)^3} \geq 1$ est vérifiée pour $n > 2.6$ et nous savons déjà que $(\frac{n-1}{n})^n \geq (\frac{n-2}{n-1})^{n-1}$ d'après le cas déjà traité $j = 0$.

Induction : Supposons que $1 \leq j < n-1$ et $n \cdot b_j^n \geq (n-1) \cdot b_j^{n-1}$ est vraie, et prouvons l'inégalité pour $j+1$.

Nous calculons pour cela le rapport $B_j^n = b_{j+1}^n / b_j^n$:

$$\begin{aligned} B_j^n &= \frac{\binom{n}{j+1}\left(\frac{1}{n}\right)^{j+1}\left(1-\frac{1}{n}\right)^{n-j-1}}{\binom{n}{j}\left(\frac{1}{n}\right)^{j}\left(1-\frac{1}{n}\right)^{n-j}} \\ &= \frac{n-j}{j+1} \cdot \frac{1}{n} \cdot \frac{n}{n-1} = \frac{n-j}{(j+1)(n-1)} \end{aligned}$$

Si on compare B_j^n et B_j^{n-1} en calculant à leur tour leur rapport, on trouve :

$$\frac{B_j^n}{B_j^{n-1}} = \frac{n-j}{(j+1)(n-1)} \cdot \frac{(j+1)(n-2)}{n-1-j} = \frac{(n-2)(n-j)}{(n-1)(n-1-j)}$$

En développant ce rapport, nous obtenons $B_j^n / B_j^{n-1} \geq 1 \Leftrightarrow j \geq 1$. Remarquons finalement que $n \cdot b_{j+1}^n \geq (n-1) \cdot b_{j+1}^{n-1} \Leftrightarrow n \cdot B_j^n \cdot b_j^n \geq (n-1) \cdot B_j^{n-1} \cdot b_j^{n-1}$.

Par hypothèse d'induction et comme $B_j^n \geq B_j^{n-1}$ pour $j \geq 1$, le résultat suit.

□

Proposition 4.2. *La procédure* BIN1 *retourne j avec probabilité b_j^n.*

Preuve Comme $\sum_{j=0}^{m} p_j^m = m \cdot \sum_{j=0}^{m} b_j^m - (m-1) \cdot \sum_{j=0}^{m} b_j^{m-1} = m - (m-1) = 1$ et en appliquant le lemme 4.1, nous obtenons que les p_j^m forment effectivement une distribution de probabilité sur $[\![0, m]\!]$.

Dénotons c_j^n la probabilité que la valeur retournée par Bin1 soit j. Nous prouvons la proposition en raisonnant par induction sur n.

Pour le cas de base $n = 1$, $c_0^1 = p_0^1 = b_0^1 = 0$ et $c_1^1 = p_1^1 = b_1^1 = 1$ comme une uniforme sur $[\![1]\!]$ retourne évidement toujours 1.

Supposons à présent que l'on ait $c_j^{n-1} = b_j^{n-1}$ pour $n \geq 2$ et tout $j \leq n-1$.

Nous obtenons alors :

$$\begin{aligned} c_j^n &= \sum_{m=1}^{n} \frac{1}{n} \cdot p_j^m = \frac{1}{n}\left(\sum_{m=1}^{n-1} p_j^m + p_j^n\right) = \frac{1}{n}\left(\left(\sum_{m=1}^{n-1} \frac{1}{n-1} p_j^m\right) \cdot (n-1) + p_j^n\right) \\ &= \frac{1}{n}\left(c_j^{n-1} \cdot (n-1) + p_j^n\right) = \frac{1}{n}\left(b_j^{n-1} \cdot (n-1) + n \cdot b_j^n - (n-1) b_j^{n-1}\right) \\ &= b_j^n \end{aligned}$$

en utilisant notre hypothèse d'induction à la seconde ligne d'équation.

□

4.1.2 Simulation de la loi Bin$(n, k/n)$

Il est aisé de remarquer que la procédure présentée dans le paragraphe précédent ne se généralise pas au cas $k \geq 2$. Autrement dit, si on pose $p_j^{k,m} = m \cdot b_j^{k,m} - (m-1) \cdot b_j^{k,m-1}$, certaines valeurs potentielles de k, m et j rendent le coefficient $p_j^{k,m}$ négatif et d'autres le rendent supérieur à 1, par exemple, nous obtenons $p_2^{2,3} = 3 \cdot b_2^{2,3} - 2 \cdot b_2^{2,2} = -2/3$.

D'une manière plus générale, un algorithme $\mathcal{A}$ simulant Bin$(n, k/n)$ qui n'aurait comme seule information sur n, une et une seule variable aléatoire X_n distribuée uniformément sur $[\![n]\!]$, n'est pas envisageable. Comme n peut être égal à k, un tel algorithme est forcé de retourner k lorsque $X_n \leq k$, d'où $\mathbb{P}(\mathcal{A}(X_n) = k) \geq k/n$. Lorsque $n = k+1$, cette probabilité est d'au moins $1/2$ dès lors que $k \geq 2$. Or, la probabilité de retourner k dans un algorithme correct est $b_k^{k,k+1} = \left(\frac{k}{k+1}\right)^k < 1/2$ pour $k \geq 2$.

Un algorithme $\mathcal{A}$ simulant Bin$(n, k/n)$ n'est toujours pas possible lorsque la variable aléatoire fournie X_n est conditionnée à être supérieure à k. Dans une telle situation, $\mathbb{P}(\mathcal{A}(X_n) = k) \geq \frac{1}{n-(k-1)}$ car l'algorithme n'a qu'un unique comportement possible lorsque $X_n = k$: retourner k. Lorsque $n = k+1$, cette probabilité est exactement de $1/2$, et donc toujours strictement supérieure à la probabilité visée.

Il se trouve que seul le cas $n = k+1$ pose réellement problème, comme le démontre le lemme suivant :

Lemme 4.3. *Pour $k \geq 1$, $n \geq k+2$, $j \in [\![0, n]\!]$,*

$$(n+1-k) \cdot b_j^{k,n} \geq (n-k) \cdot b_j^{k,n-1}.$$

Preuve Remarquons que le cas $j = n$ est trivial, et pour $j \in [\![0, n-1]\!]$, nous pouvons reformuler l'inégalité sous la forme $A(j) \geq 1$ où :

$$A(j) = \frac{(n+1-k) \cdot b_j^{k,n}}{(n-k) \cdot b_j^{k,n-1}} = \frac{n+1-k}{n-j} \left(\frac{n-k}{n-1-k}\right)^{n-1-j} \left(\frac{n-1}{n}\right)^{n-1}.$$

La suite de la preuve consistera à prouver dans un premier temps le cas $A(k) \geq 1$, puis dans un second temps à montrer que $A(k)$ est un des minima de $A(j)$.

Pour $j = k$, en notant $a = n - k$ (donc $a \geq 2$), $A(k) \geq 1$ se réécrit :

$$\frac{a+1}{a} \left(\frac{a}{a-1}\right)^{a-1} \left(\frac{a+k-1}{a+k}\right)^{a+k-1} \geq 1.$$

Comme $\frac{a+1}{a} \left(\frac{a}{a-1}\right)^{a-1}$ et $\left(\frac{a+k-1}{a+k}\right)^{a+k-1}$ sont décroissantes lorsque $a \geq 2$, et convergent respectivement vers e et $1/e$, on obtient $A(k) \geq 1$.

De plus, pour tout $j \in [\![0, n-2]\!]$, comme $A(j+1)/A(j) = \frac{n-j}{n-j-1} \frac{n-k-1}{n-k}$, on a $A(j+1) \geq A(j)$ si et seulement si $j \geq k$. Donc $A(k) = A(k+1)$ sont les minima de $A(j)$. □

Le lemme précédent et une adaptation de la preuve de la proposition 4.2 montre que, sous la garantie $n \geq k+2$ (qui plus forte que notre contrainte initiale de $n \geq k$), une unique uniforme X_n sur $[\![k, n]\!]$ suffit pour simuler la distribution binomiale ciblée en retournant j avec probabilité $p_j^{k,m} = (m-k+1) \cdot b_j^{k,m} - (m-k) \cdot b_j^{k,m-1}$ lorsque $X_n = m$.

Nous proposons l'algorithme suivant qui ne suppose aucune garantie sur n, mis à part $n \geq k$, et utilisant au maximum 2 appels à un générateur d'uniformes sur $[\![k, n]\!]$.

Algorithme 19 BIN-K

1. Générer une première uniforme m sur $[\![k, n]\!]$; si $m = k$, et uniquement dans ce cas, remplacer m par une nouvelle uniforme sur $[\![k, n]\!]$, puis passer à l'étape 2.
2. Renvoyer enfin $0 \leq j \leq n$, pour $j \neq k$, avec probabilité

$$p_j^{k,m} = \frac{(m-k+1)^3 \cdot b_j^{k,m} - (m-k+2)(m-k)^2 \cdot b_j^{k,m-1}}{(m-k+2)(m-k+1)}$$

et k avec probabilité complémentaire, *i.e.* $p_k^{k,m} = 1 - \sum_{j=0, j\neq k}^{m} p_j^{k,m}$.

Après l'étape 1 de l'algorithme, la probabilité $\ell_m^{k,n}$ d'obtenir m est

$$\ell_m^{k,n} = \begin{cases} \dfrac{1}{(n+1-k)^2} & \text{si } m = k \\ \dfrac{n-k+2}{(n+1-k)^2} & \text{si } m \neq k \end{cases}$$

car la seule possibilité d'obtenir k est de tomber deux fois dessus, d'où $\ell_k^{k,n} = 1/(n-(k-1))^2$; pour $m \neq k$, $\ell_m^{k,n} = 1/(n-(k-1)) + 1/(n-(k-1))^2 = (n-k+2)/(n-k+1)^2$ car m est obtenu soit au premier essai, soit après avoir obtenu la valeur k au premier tirage. Pour $n \geq k+1$, remarquons que le rapport $\ell_m^{k,n}/\ell_m^{k,n-1} = \frac{(n-k+2)(n-k)^2}{(n-k+1)^3}$ ne dépend pas de m.

Proposition 4.4. Bin-k *retourne j avec probabilité $b_j^{k,n}$.*

Preuve La preuve utilise des arguments analogues à celle de la proposition 4.2 : nous prouvons en premier lieu que les $(p_j^{k,m})_{j\in[\![0,m]\!]}$, pour $1 \leq k \leq m$, forment effectivement une distribution de probabilité sur $[\![0,m]\!]$, et ensuite que la valeur retournée par l'algorithme est bien distribuée selon la loi $\mathrm{Bin}(n,k/n)$.

Nous commençons donc par montrer que les $p_j^{k,m}$ se somment à 1 et qu'ils sont tous positifs.

Le premier point est évident par définition de $p_k^{k,m}$.

Pour le second point, en ce qui concerne les entiers k, n et j couvert par le lemme 4.3, nous avons $p_j^{k,n} \geq 0$ pour $j \neq k$ en remarquant $(n+1-k)^2 = (n+2-k)(n-k)+1$; il suffit en effet de multiplier l'inégalité du lemme par $(n+1-k)^2 \geq (n+2-k)(n-k)$ car nous ne manipulons que des nombres positifs, y compris le dénominateur de $p_j^{k,n}$. De plus, en notant que pour les deux cas restants $n=k$ et $n=k+1$, la positivité est évidente pour $j \neq k$ car $b_j^{k,k} = b_j^{k,k-1} = 0$, il nous reste uniquement à prouver la positivité des $p_k^{k,m}$ pour $m \geq k$; $p_k^{k,m}$ se réécrit en notant $\alpha_m = (m-k+1)^2/(m-k+2)$:

$$\begin{aligned} p_k^{k,m} &= 1 - \sum\nolimits_{j=0,j\neq k}^{m} p_j^{k,m} = 1 - \sum\nolimits_{j=0,j\neq k}^{m} \left(\alpha_m b_j^{k,m} - \alpha_{m-1} b_j^{k,m-1}\right) \\ &= 1 - \alpha_m \sum\nolimits_{j=0,j\neq k}^{m} b_j^{k,m} + \alpha_{m-1} \sum\nolimits_{j=0,j\neq k}^{m} b_j^{k,m-1} \\ &= 1 - \alpha_m(1 - b_k^{k,m}) + \alpha_{m-1}(1 - b_k^{k,m-1}) \\ &= 1 - \alpha_m + \alpha_m b_k^{k,m} + \alpha_{m-1} - \alpha_{m-1} b_k^{k,m-1}. \end{aligned}$$

Or $1 - \alpha_m + \alpha_{m-1} = \frac{1}{(m-k+2)(m-k+1)} \geq 0$. Enfin $\alpha_m b_k^{k,m} - \alpha_{m-1} b_k^{k,m-1} \geq 0$ lorsque $m \geq k+2$ (d'après le lemme 4.3 et les arguments du début de la preuve) et $m = k$ car $b_k^{k,k-1} = 0$.

Ainsi, l'unique cas restant à montrer est $p_k^{k,k+1} \geq 0$, que l'on réécrit

$$p_k^{k,k+1} = 1 - \alpha_{k+1} + \alpha_k + \alpha_{k+1} b_k^{k,k+1} - \alpha_k b_k^{k,k} = \frac{4 \cdot b_k^{k,k+1} - 1}{3}.$$

Or $b_k^{k,k+1} = (k/(k+1))^k > 1/e$, d'où $p_k^{k,k+1} > 0{,}15$.

Il ne reste plus qu'à calculer $c_j^{k,n}$ la probabilité que l'algorithme retourne j, pour $j \in [\![0,n]\!]$.

Nous travaillons par induction sur $n \geq k+1$ pour $j \neq k$, avec comme cas de base $n = k$ donnant $c_j^{k,k} = 0$:

$$\begin{aligned} c_j^{k,n} &= \sum_{m=k}^{n} \ell_m^{k,n} \cdot p_j^{k,m} = \sum_{m=k}^{n-1} \left(\frac{\ell_m^{k,n}}{\ell_m^{k,n-1}} \ell_m^{k,n-1} p_j^{k,m} \right) + \ell_n^{k,n} p_j^{k,n} \\ &= \frac{(n-k+2)(n-k)^2}{(n-k+1)^3} \cdot \sum_{m=k}^{n-1} \ell_m^{k,n-1} p_j^{k,m} + \ell_n^{k,n} p_j^{k,n} \\ &= \frac{(n-k+2)(n-k)^2}{(n-k+1)^3} \cdot c_j^{k,n-1} + \frac{n-k+2}{(n-k+1)^2} p_j^{k,n} \end{aligned}$$

En utilisant notre hypothèse d'induction, nous obtenons

$$c_j^{k,n} = \frac{(n-k+2)(n-k)^2}{(n-k+1)^3} \cdot b_j^{k,n-1} + b_j^{k,n} - \frac{(n-k+2)(n-k)^2}{(n-k+1)^3} b_j^{k,n-1} = b_j^{k,n}.$$

Comme seules des valeurs entre 0 et n peuvent être renvoyées, on retrouve

$$c_k^{k,n} = 1 - \sum_{j=0, j\neq k}^{n} c_j^{k,n} = b_k^{k,n}.$$

□

Remarque BIN-K utilise en moyenne $1 + \frac{1}{n-k+1} = 1 + \mathcal{O}(1/n)$ tirages de variables uniformes sur $[\![k, n]\!]$. De plus, si des tirages d'uniformes sur $[\![n]\!]$ sont utilisées afin de simuler des uniformes sur $[\![k, n]\!]$, l'algorithme en utilisera en moyenne $1+\mathcal{O}(1/n)$ par simulation de Unif($[\![k, n]\!]$), donc au total $1 + \mathcal{O}(1/n)$ tirages sur $[\![n]\!]$ seront utilisées.

4.1.3 Simulation d'uniformes à partir de RV

Nous terminons cette section en décrivant comment des uniformes sur $[\![n]\!]$ peuvent être simulées, dans notre modèle original, *i.e.* à partir d'appels à un générateur d'uniformes sur un ensemble V de taille n comme notre primitive RV.

Pour cela, il suffit d'appeler la primitive RV jusqu'à ré-obtenir le premier élément renvoyé par la fonction, puis de retourner le nombre d'éléments *différents* vus au cours de la procédure.

Cette méthode simule la construction du cycle contenant un élément donné (le premier obtenu via RV) dans une permutation uniforme de V. Lorsque des valeurs déjà rencontrées, autre que la première, sont tirées, elle sont simplement ignorées, comme lorsque l'on génère le cycle de l'élément 1 avec des rejets en tirant dans $[\![n]\!]$.

Il est simple de vérifier que cette taille est uniformément distribuée sur l'ensemble $[\![n]\!]$: le nombre de permutations de taille n où un élément fixé est contenu dans un cycle de taille i est

$$\binom{n-1}{i-1} \cdot (i-1)! \cdot (n-i)! = (n-1)! = \frac{1}{n} \cdot n!.$$

En espérance, il faudra $n+1$ appels à `RV` afin de générer une telle variable uniforme sur $[\![n]\!]$.

De plus, n'importe quel algorithme simulant la loi uniforme sur $[\![n]\!]$ aura besoin en moyenne de $\Omega(n)$ appels à `RV`. En effet, il faut au minimum i tirages avant de pouvoir renvoyer $i \in [\![n]\!]$, car un algorithme renvoyant $\ell > i$ après seulement i tirages est nécessairement incorrect : si au plus i valeurs distinctes ont été observées, il est possible que n soit égal à i et ℓ ne fait pas partie du support de la loi Unif($[\![n]\!]$). Or, un algorithme qui tire une uniforme sur $[\![n]\!]$ a probabilité supérieure à $1/2$ de retourner une valeur d'au moins $n/2$ ce qui fournit la borne $\Omega(n)$ sur l'espérance.

L'algorithme présenté est donc optimal à une constante multiplicative près.

4.2 Algorithme du premier doublon

Comme il est montré dans le paragraphe qui précède, simuler la loi binomiale dans notre modèle `RV` en utilisant les algorithmes asymptotiquement optimaux en tirages d'uniformes, présentés au cours de la section 4.1, coûte en moyenne $n+1$ appels à la primitive `RV` pour le cas $k=1$, et $n+1+\mathcal{O}(1/n)$ pour les autres valeurs de k. Utiliser ces algorithmes de simulation tel quels lors de maintenance de graphes aléatoires va complètement à l'encontre de notre politique de mise à jour efficace.

Il est en fait possible d'améliorer cette complexité et d'obtenir un coût sous-linéaire. Dans cette section, nous présentons un nouvel algorithme simulant la loi $\mathrm{Bin}(n, 1/n)$, appelé algorithme du premier doublon, qui nécessite en moyenne $\mathcal{O}(\sqrt{n})$ appels à un générateur d'uniformes sur V avec $|V| = n$. Ici, nous entendons par *doublon*, un élément qui est obtenu une deuxième fois par le générateur.

Comme pour nos précédentes procédures, l'algorithme se présente en deux étapes, où les tirages sur V n'interviennent que dans la résolution de la première étape.

Algorithme 20 Algorithme-Premier-Doublon

1. Tirer successivement des éléments uniformes sur V, jusqu'à obtenir un des éléments une seconde fois.
2. Retourner le nombre de points fixes d'une permutation aléatoire uniforme des éléments tirés.

Théorème 4.5. *Pour tout $j \in [\![0, n]\!]$, l'algorithme du premier doublon retourne j avec probabilité b_j^n.*

Preuve Soit $U \subset V$ avec $|U| = j$, $D \subset V$ l'ensemble des éléments retenus après l'étape 1, $L = |D|$ et σ la permutation obtenue après l'étape 2.

Nous définissons les évènements suivants $A_u := \{u \text{ est point fixe dans } \sigma\}$, $A_U = \bigcap_{u \in U} A_u$ et $B_U := \{U \subset D\}$.

Notons dans un premier temps que pour $1 \leq \ell \leq n$,

$$\mathbb{P}(L = \ell) = \left(\prod_{i=0}^{\ell-1} \frac{n-i}{n}\right) \cdot \frac{\ell}{n} = \frac{n!}{(n-\ell)!} \frac{\ell}{n^{\ell+1}}.$$

De plus pour $\ell \geq j$, nous avons

$$\mathbb{P}(B_U \mid L = \ell) = \frac{\binom{n-j}{\ell-j}}{\binom{n}{\ell}} = \frac{(n-j)!\,\ell!}{(\ell-j)!\,n!}$$

car conditionnellement à $L = \ell$, D est un ℓ-sous-ensemble uniforme de V.

Enfin, conditionné à B_U et $L = \ell$, la probabilité de A_U est $(\ell-j)!/\ell!$ si $\ell \geq j$, et 0 si $\ell < j$.

Nous pouvons à présent calculer la probabilité de l'évènement A_U :

$$\begin{aligned}
\mathbb{P}\left(\bigcap_{u\in U} A_u\right) &= \sum_{\ell=1}^{n} \mathbb{P}\left(A_U \mid B_U, L = \ell\right) \cdot \mathbb{P}(B_U \mid L = \ell) \cdot \mathbb{P}(L = \ell) \\
&= \sum_{\ell=j}^{n} \frac{(\ell-j)!}{\ell!} \cdot \frac{(n-j)!\,\ell!}{(\ell-j)!\,n!} \cdot \frac{n!\,\ell}{n^{\ell+1} \cdot (n-\ell)!} \\
&= \sum_{\ell=j}^{n} \frac{(n-j)!}{(n-\ell)!} \cdot \frac{\ell}{n^{\ell+1}} = \frac{1}{n^j} \cdot \sum_{\ell=j}^{n} \underbrace{\frac{(n-j)!}{(n-\ell)!} \frac{\ell}{n^{\ell-j+1}}}_{\mathbb{P}(L=\ell \mid L\geq j)} \\
&= \left(\frac{1}{n}\right)^j \sum_{\ell\geq j} \mathbb{P}(L = \ell \mid L \geq j) = \left(\frac{1}{n}\right)^j
\end{aligned}$$

Donc l'ensemble $\{A_u\}_{u\in V}$ est mutuellement indépendant.

Nous pouvons enfin conclure que les indicatrices $\mathbb{1}_{A_u}$ sont des Bernoulli indépendantes, de paramètre $1/n$, donc que leur somme est une binomiale de paramètre n et $1/n$.

□

Si au lieu de renvoyer le nombre de points de la permutation générée, on retourne l'ensemble de ces points fixes, on obtient alors un *sous-ensemble binomial* (défini ci-dessous) de probabilité $1/|V|$ de V, un avantage que l'on exploitera par la suite.

Définition 4.1. Un *sous-ensemble binomial* de probabilité p d'un ensemble V, est un sous-ensemble aléatoire de V dont chaque élément $v \in V$ a probabilité p d'y être présent, indépendamment des autres.

Proposition 4.6. *L'algorithme 20 utilise en moyenne $\mathcal{O}(\sqrt{|V|})$ tirages d'uniformes sur V, et prend un temps moyen de $\mathcal{O}(\sqrt{|V|})$.*

Preuve Le coût moyen de l'algorithme, en tirages d'uniformes sur V, correspond au nombre moyen de tirages avant d'obtenir un doublon (la deuxième partie n'utilisant que de l'aléa *local*). Il est connu que ce nombre est en moyenne $\sqrt{\frac{\pi}{2}n} + \mathcal{O}(1/\sqrt{n})$ d'après le célèbre *paradoxe des anniversaires* (voir par exemple [67, 50]).

La complexité, en temps de calcul, est quant à elle linéaire par rapport à ce nombre moyen de tirages (en utilisant un générateur uniforme de permutations, linéaire en moyenne).

□

Cet algorithme peut de plus être simulé en utilisant une construction combinatoire particulière, qui nous permettra de passer d'un coût moyen d'environ $1{,}25\sqrt{n}$ à $\mathcal{O}(1)$. Cette construction, présentée dans la section suivante, travaillera sur les permutations afin d'anticiper le nombre de points fixes de la permutation finale. Le point clé est de lier le nombre de points fixes d'une permutation uniforme, et la génération aléatoire uniforme de permutations.

Dans la section 4.4 concluant ce chapitre, nous présentons une méthode afin de simuler la loi $\mathrm{Bin}(n,\, k/n)$ en appelant $\mathcal{O}(k)$ fois une procédure générant un ensemble binomial de probabilité $1/|V|$ comme l'algorithme du premier doublon présenté dans cette section.

4.3 Un nouvel arbre de génération pour les permutations

Dans cette section, nous décrivons un nouvel arbre de génération uniforme pour les permutations. Notre arbre a la propriété spécifique que, pour la plupart ($n! - 2^{n-1}$ pour être précis) des permutations, tous leurs descendants dans l'arbre possèdent le même nombre de points fixes. Notre arbre est optimal en ce sens, c'est-à-dire qu'un maximum de permutations *préserveront* leur nombre de points fixes dans leur descendance. Dans la section suivante, nous montrons comment cette propriété peut être exploitée afin de simuler la distribution $\mathrm{Bin}(n,\, k/n)$ dans notre contexte.

Nous profiterons de cette section pour exhiber deux autres applications possibles de notre arbre. La première est un nouvel algorithme de génération pour les dérangements, qui nécessite en moyenne $n + \mathcal{O}(1)$ appels à un générateur de nombres aléatoires. La seconde est une méthode combinatoire de simulation de la loi de Poisson de paramètre 1.

4.3.1 Points fixes dans les permutations

Pour chaque permutation $\sigma \in \mathcal{S}_n$, on note $\mathrm{fp}(\sigma)$ son nombre de points fixes, *i.e.* $\mathrm{fp}(\sigma) = |\{i \leq n \mid \sigma(i) = i\}|$. Les permutations qui n'ont aucun point fixe sont appelées *dérangements*. L'ensemble des dérangements sur $[\![n]\!]$ est noté $\mathcal{D}_n$.

Dans ce qui suit, nous manipulerons des permutations uniformes sur $\mathcal{S}_n$. Il est bien connu que dans une permutation uniforme de taille n, chaque élément a probabilité $(n-1)!/n! = 1/n$ d'être point fixe. Donc, par linéarité de l'espérance, le nombre moyen de points fixes dans une permutation uniforme est toujours exactement 1, et ce, quelque soit la taille considérée.

Il est légèrement moins connu, que le nombre de points fixes dans une permutation aléatoire uniforme de taille n suit une distribution qui, lorsque n tend vers l'infini, converge vers la loi de Poisson de paramètre 1.

Une conséquence probabiliste est que la convergence en distribution peut être réalisée de façon presque sûre. Autrement dit, on peut construire dans un même espace de probabilité, une séquence $(\sigma_n)_{n\geq 1}$ où σ_n est uniforme sur $\mathcal{S}_n$, de telle sorte qu'avec probabilité 1, $\lim_{n\to\infty} \mathrm{fp}(\sigma_n)$ existe et est distribuée comme Poi(1). Notre arbre peut être vu, dans une certaine mesure, comme une construction combinatoire réalisant cette convergence.

Notation Nous utiliserons deux représentations pour décrire une permutation σ de $[\![n]\!]$ dans les exemples :

- comme un mot w de longueur n, où chaque élément de $[\![n]\!]$ n'apparaît qu'une seule fois, et où la i-ème lettre w_i correspond à $\sigma(i)$ l'image de i par la permutation σ ; par exemple $\sigma = 15342$ est une permutation de $\{1, \ldots, 5\}$.
- comme une liste de cycles où chaque cycle est noté entre parenthèses. Par convention, les cycles sont écrits en commençant par leur plus petit élément et la liste des cycles est triée selon l'ordre naturel du plus petit élément des cycles ; par exemple, la permutation $\sigma = 15342$ se note $(1)(25)(3)(4)$. Il sera parfois plus aisé de voir la permutation $\sigma \in \mathcal{S}_n$ comme un graphe dirigé sur $[\![n]\!]$ où l'arc (u,v) est présent si et seulement si $\sigma(u) = v$, ce qui correspond à un graphe dirigé où chaque composante connexe est un cycle : on parle alors de représentation sous forme d'ensemble de cycles.

Remarque Une seule écriture est possible pour $\sigma \in \mathcal{S}_n$ dans chacune des notations.

4.3.2 Arbres de génération classiques pour les permutations

Définition 4.2. Un arbre de génération (uniforme) pour les permutations est un arbre infini dont le n-ème niveau est l'ensemble des permutations de taille n, et dont chaque nœud au niveau n, possède exactement $n+1$ enfants ordonnés.

Nous fixons la racine [1] d'un tel arbre à l'unique permutation de taille 1.

Un arbre de génération est équivalent à la description d'une bijection ϕ_n, pour chaque $n \geq 1$, entre $\mathcal{S}_n \times [\![n+1]\!]$ et $\mathcal{S}_{n+1}$, où $\sigma' = \phi_n(\sigma, i)$ est le i-ème enfant de la permutation $\sigma \in \mathcal{S}_n$ dans l'arbre.

1. L'unique permutation de taille 0 peut aussi être prise en racine, et possède alors un unique enfant : la permutation de taille 1. Cela ne change en rien les propriétés de l'arbre.

Nous dirons que σ est le *père* de σ' ; l'ensemble des permutations contenues dans le sous-arbre enraciné en σ est appelé *descendants* de σ.

De plus, un arbre de génération fournit également un algorithme de génération uniforme pour les permutations : à partir d'une permutation σ de taille n choisie selon la loi uniforme et d'un entier $i \in [\![n+1]\!]$, indépendant de σ, on obtient une permutation uniforme de taille $n+1$ en récupérant simplement le i-ème enfant de σ dans l'arbre. Ainsi, si en partant de la racine, on choisit une *branche* uniforme à chaque niveau jusqu'à atteindre le niveau n, alors la permutation obtenue à la fin de cette *descente aléatoire* dans l'arbre, sera bien uniforme sur $\mathcal{S}_n$.

Remarque Dans cette section, nous allons décrire un arbre de génération dont la propriété essentielle est qu'à partir de n'importe quel nœud de l'arbre, une descente aléatoire atteindra avec probabilité 1 une permutation à partir de laquelle le nombre de points fixes de tous ses descendants est identique. Autrement dit, le nombre de points fixes d'une descente aléatoire infinie est *figé* après un certain temps avec probabilité 1.

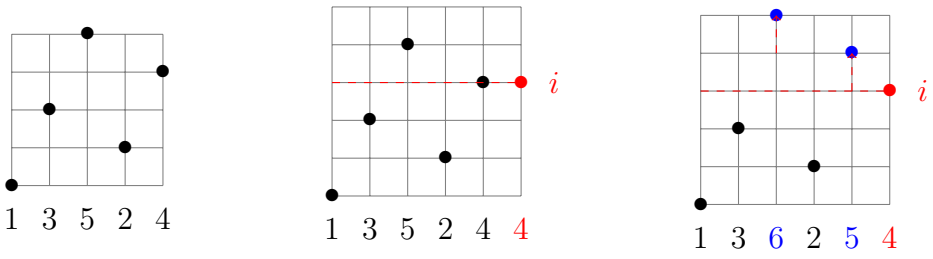
Il est possible de formuler beaucoup de descriptions combinatoires d'arbres de génération. Pour servir d'exemples, nous décrivons rapidement deux d'entre elles et comparons leur *préservation* du nombre de points fixes.

Les premiers niveaux de ces arbres sont illustrés figure 4.14. Dans cette figure et celles qui vont suivre, les enfants sont dessinés de manière ascendante (le premier enfant est celui le plus bas, le dernier est celui le plus haut), les points fixes sont soulignés et les permutation en gras n'ont pas le même nombre de points fixes que leur père dans l'arbre.

Insertion par décalage

Dans l'arbre de génération d'*insertion par décalage* σ', le i-ème enfant de la permutation $\sigma \in \mathcal{S}_n$, est construit à partir de σ en assignant i comme image à $n+1$ et en augmentant les images aussi grandes que i d'une unité ; *i.e.* $\sigma'(n+1) = i$ et pour tout $1 \leq j \leq n$, $\sigma'(j) = \sigma(j)$ si $\sigma(j) < i$ et $\sigma'(j) = \sigma(j) + 1$ si $\sigma(j) \geq i$.

Exemple 4.1. La transformation est particulièrement lisible en dessinant les permutations sous forme de diagramme. Par exemple, sur l'illustration suivante, on construit le 4-ème enfant de la permutation 13524 : 136254.



■

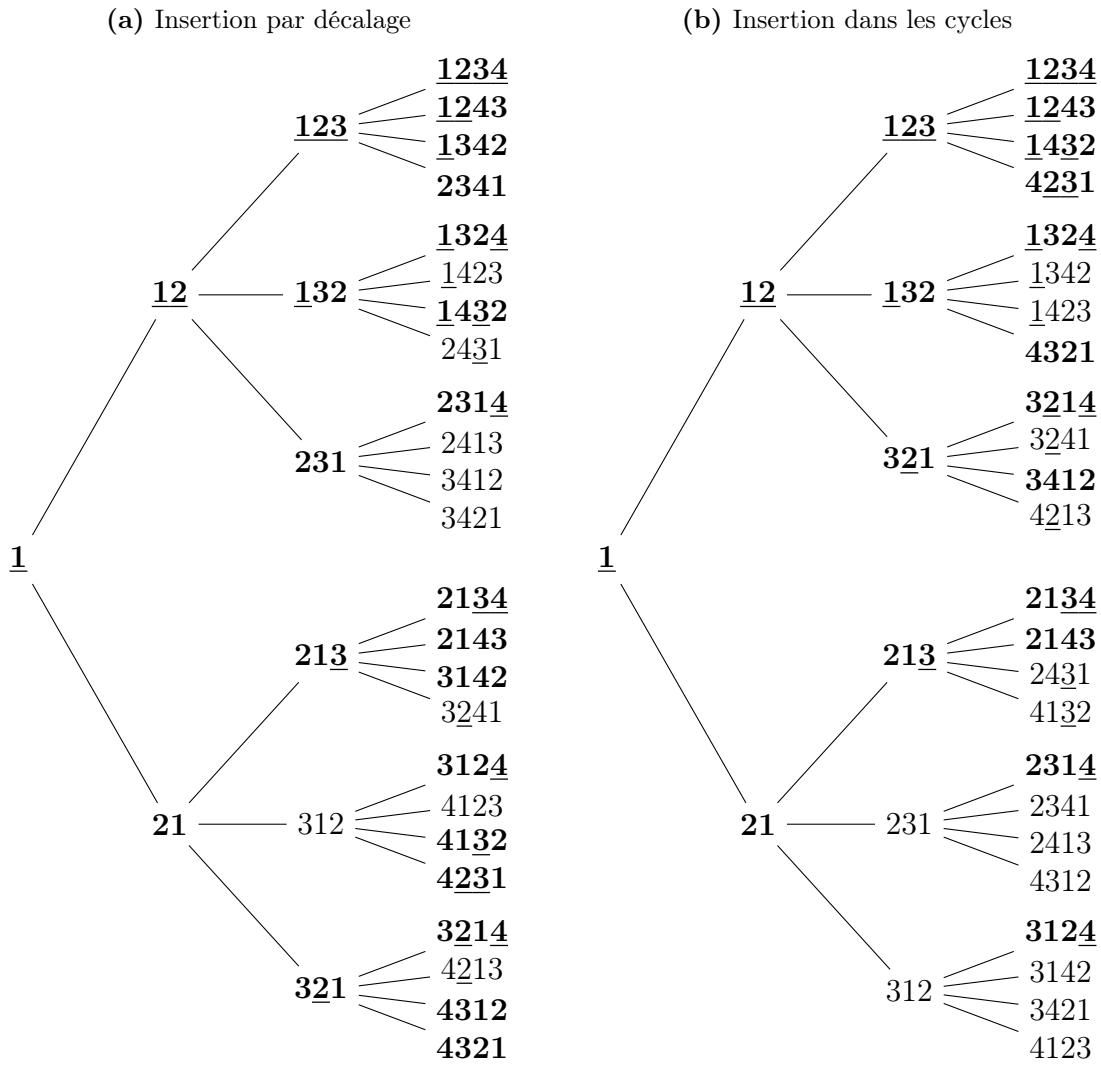


FIGURE 4.14 – Quatre premiers niveaux d'arbres de génération classiques.

Dans l'arbre de génération d'insertion par décalage, le nombre de points fixes peut changer drastiquement entre deux niveaux de l'arbre. Par exemple, le premier enfant de la permutation identité $12\dots n$ ne possède pas de points fixes (diminution de n points fixes) et le premier enfant du cycle décroissant $n12\dots n-1$ possède $n-1$ points fixes (augmentation de $n-1$ points fixes).

De plus, le nombre de permutations *changeantes*, c'est-à-dire dont le père dans l'arbre possède un nombre de points fixes différent, est important : au moins $(n-1)!$ au niveau n, car chaque permutation a au minimum un enfant (son dernier) avec un nombre de points fixes distinct d'elle-même.

La propriété essentielle de notre arbre (décrite dans la remarque du début de section page 119) n'est en outre pas vérifiée. Partant de n'importe quelle permutation, une descente infinie changera le nombre de points fixes une infinité de fois avec

probabilité 1 : le $(n+1)$-ème enfant a toujours exactement 1 point fixe de plus, et une descente aléatoire le sélectionne avec probabilité $\frac{1}{n+1}$ (indépendamment de la permutation) ; comme la série $\sum_n \frac{1}{n+1}$ diverge, le second lemme de Borel-Cantelli implique que presque toute descente aléatoire infinie sélectionne infiniment souvent le dernier enfant.

Insertion dans les cycles

L'arbre de génération, dit d'*insertion dans les cycles*, consiste à *insérer* le nouvel élément entre i et son image, *i.e.* σ', le i-ème enfant de $\sigma \in \mathcal{S}_n$ est donnée par : si $i = n+1$, alors $\sigma'(n+1) = n+1$ et, pour $1 \leq j \leq n$, $\sigma'(j) = \sigma(j)$; sinon $i \neq n+1$, on pose alors $\sigma'(i) = n+1$, $\sigma'(n+1) = \sigma(i)$, et $\sigma'(j) = \sigma(j)$ pour toutes les autres valeurs de j.

Exemple 4.2. Comme son nom l'indique, la transformation est facilement visible lorsqu'on représente les permutations sous forme d'ensemble de cycles.

Par exemple, la permutation de droite sur le schéma ci-dessous est obtenue en tant que 10-ème enfant, du 5-ème enfant, du second enfant de $(1354)(2)(67)$:

■

Contrairement à l'arbre de génération de la section précédente, le nombre de points fixes évolue peu entre une permutation et ses enfants. Au maximum, un point fixe est retiré (lorsque i est un point fixe) ou un point fixe est ajouté (lorsque $i = n+1$), et dans les autres cas, le nombre de points fixes ne change pas.

Le nombre de permutations changeantes est exactement $2(n-1)!$ au niveau n. Comme pour l'arbre précédent, une descente infinie dans l'arbre rencontrera des permutations changeantes une infinité de fois avec probabilité 1.

4.3.3 Notre arbre de génération

Nous présentons ici notre arbre de génération pour les permutations.

Propriétés

L'arbre de génération que nous allons décrire possède les propriétés suivantes :

— Il y a exactement 2^{n-1} permutations changeantes au niveau n, c'est-à-dire que 2^{n-1} permutations de taille n ont un nombre de points fixes différent de leur père [2] dans l'arbre et les autres $n! - 2^{n-1}$ ne sont pas changeantes.

2. Bien que la racine n'ait pas de père, pour des raisons pratiques, elle aussi, est considérée changeante.

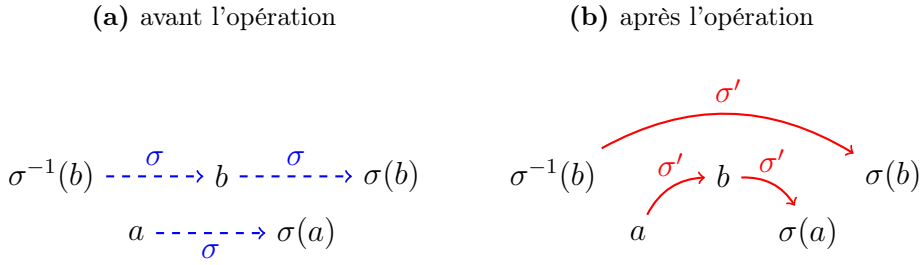


FIGURE 4.15 – Transformation de σ en σ' par l'opération insérer b après a.

— Si une permutation n'est pas changeante (même nombre de points fixes que son père), alors tous ses descendants dans l'arbre ont le même nombre de points fixes. Autrement dit, la propriété d'être *non-changeante* est héréditaire.
— Lorsqu'une permutation σ est changeante, ses enfants ont nécessairement le même nombre de points fixes $\mathrm{fp}(\sigma)$ que leur père, ou $\mathrm{fp}(\sigma) \pm 1$. De plus, exactement un de ces enfants a $\mathrm{fp}(\sigma) + 1$ points fixes (mais potentiellement plusieurs peuvent avoir $\mathrm{fp}(\sigma) - 1$ points fixes).

Opération « insérer après »

Notre arbre de génération peut se décrire entièrement à partir d'une opération simple sur les permutations appelée *insérer après*. L'opération « insérer b après a » pour $a \neq b$, envoie une permutation de taille n sur une autre de même taille, et consiste à enlever l'élément b de son cycle, puis l'*insérer* dans le cycle de a, juste après l'élément a (voir figure 4.15).

Précisément, la permutation $\sigma' \in \mathcal{S}_n$ obtenue après l'opération insérer b après a appliquée à $\sigma \in \mathcal{S}_n$ est donnée par :

— Si b n'est pas un point fixe de σ, $\sigma'(\sigma^{-1}(b)) = \sigma(b)$;
— $\sigma'(a) = b$ et $\sigma'(b) = \sigma(a)$;
— pour tout $i \in [\![n]\!]$, $i \notin \{\sigma^{-1}(b), a, b\}$, $\sigma'(i) = \sigma(i)$.

Exemple 4.3. Dans $(123)(4)(5)$,
— Insérer 5 après 1 donne $(1523)(4)$
— Insérer 1 après 5 donne $(15)(23)(4)$
— Insérer 5 après 4 donne $(123)(45)$
— Insérer 4 après 5 donne $(123)(45)$
— Insérer 3 après 2 donne $(123)(4)(5)$ ■

Remarque L'arbre de génération d'insertion dans les cycles présenté au § 4.3.2 peut lui aussi être décrit simplement à l'aide de cette opération. La permutation σ', i-ème enfant de $\sigma \in \mathcal{S}_n$, est obtenue en ajoutant $n+1$ comme point fixe à σ, puis si $i \neq n+1$, en insérant $n+1$ après i.

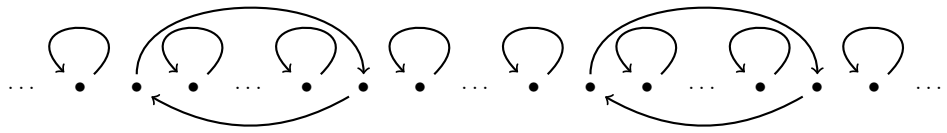


Figure 4.16 – Représentation d'une permutation spéciale.

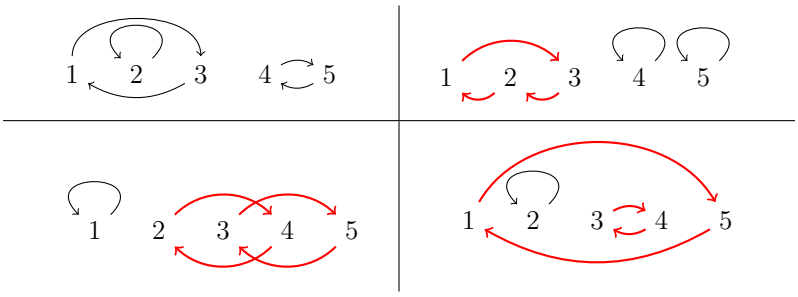
Permutations spéciales

Afin de décrire notre arbre de génération, nous définissons une classe de permutations appelées *permutations spéciales*. Ces permutations ont un rôle important dans la construction de notre arbre et correspondent exactement aux permutations changeantes de l'arbre (permutations dont le père dans l'arbre possède un nombre différent de points fixes).

Une permutation est spéciale si elle ne contient que des cycles de taille au plus 2 et si lorsque l'on dessine ses cycles de taille 2 en alignant leurs éléments par ordre croissant (voir figure 4.16), aucune paire de cycles n'est imbriquée ou chevauchante. Deux cycles de taille 2 sont dits *imbriqués* si tous les éléments de l'un sont compris *entre* les deux éléments de l'autre, *i.e.* (ab) et (cd) sont imbriqués si $a < c < d < b$ ou $c < a < b < d$. Deux cycles de taille 2 sont dits *chevauchants* si seulement un élément de l'un se retrouve *entre* ceux de l'autre, *i.e.* (ab) et (cd) sont chevauchants si $a < c < b < d$ ou $c < a < d < b$.

Définition 4.3. Une permutation σ de taille n est spéciale si pour tout $i \in [\![n]\!]$, $\sigma^2(i) = i$ et pour tout $i < j < \sigma(i)$, $\sigma(j) = j$.

Exemple 4.4. La permutation $(13)(2)(45)$ est spéciale alors que les permutations $(132)(4)(5)$, $(1)(24)(35)$ et $(15)(2)(34)$ ne le sont pas, comme on peut facilement le remarquer sur les représentations suivantes :



■

Il est facile de déterminer le nombre de permutations spéciales de taille n : il en existe autant que de sous-ensembles pairs de $[\![n]\!]$, soit 2^{n-1}. En effet, à chaque sous-ensemble $S \subset [\![n]\!]$ dont la taille est paire, on associe l'unique permutation spéciale obtenue en *couplant* les éléments de S deux par deux dans l'ordre croissant, et où les éléments restants appartenant à $[\![n]\!] \setminus S$ sont points fixes.

Ici, on entend par *coupler*, le fait de les associer dans un cycle de taille 2, par exemple à l'ensemble $\{2, 5, 6, 7\} \subset [\![7]\!]$ est associé à la permutation $(1)(25)(3)(4)(67)$.

Enfin, le nombre de sous-ensembles pairs de $[\![n]\!]$ est 2^{n-1} car il existe une involution simple qui change la parité du sous-ensemble S : si 1 appartient à S, l'enlever, sinon l'y rajouter.

Description

Notre arbre de génération est basé sur une bijection sur les dérangements due à Rankotondrajao [84]. La bijection τ_n qui y est décrite envoie un couple de $\mathcal{D}_{n-1} \times [\![n]\!] \setminus \{(\Delta_{n-1},n)\}$ vers $\mathcal{D}_n$ pour n impair, et de $\mathcal{D}_{n-1} \times [\![n]\!]$ vers $\mathcal{D}_n \setminus \{\Delta_n\}$ pour n pair. Pour n pair, le dérangement Δ_n est un dérangement particulier de taille n, appelé dérangement *critique*, qui consiste en $n/2$ cycles de taille 2 où chaque élément impair est apparié avec son successeur dans l'ordre naturel ; par exemple, $\Delta_6 = (12)(34)(56)$.

Cette bijection permet en outre de prouver combinatoirement une célèbre récurrence $d_n = nd_{n-1}+(-1)^n$ sur le nombre $d_n = |\mathcal{D}_n|$ de dérangements de taille n. Notre construction utilise une légère modification de la bijection (voir règle 4 de l'arbre) sur les dérangements [3] de certaines permutations non-spéciales. Les notations $p(\sigma)$ et $p'(\sigma)$, définies ci-après, reprennent les mêmes définitions que dans la bijection originale τ_n.

Notation Nous notons $\gamma(\sigma)$ le plus grand non-point fixe de σ (ou 0 si σ est l'identité). De plus, pour les permutations non-spéciales, on note $p(\sigma)$ le plus petit élément qui est soit dans un cycle de taille ≥ 3, soit tel qu'il existe un non-point fixe entre lui et son image [4], et $p'(\sigma)$ le plus petit non-point fixe plus grand que $p(\sigma)$.

Exemple 4.5. Reprenant les permutations présentées dans l'exemple 4.4 :

- $\sigma_a = (13)(2)(45)$: $\gamma(\sigma_a) = 5$ et comme la permutation est spéciale $p(\sigma_a)$ et $p'(\sigma_a)$ ne sont pas définis.
- $\sigma_b = (123)(4)(5)$: $p(\sigma_b) = 1$, $p'(\sigma_b) = 2$, et $\gamma(\sigma_b) = 3$.
- $\sigma_c = (1)(24)(35)$: $p(\sigma_c) = 2$, $p'(\sigma_c) = 3$, et $\gamma(\sigma_b) = 5$.
- $\sigma_d = (15)(2)(34)$: $p(\sigma_d) = 1$, $p'(\sigma_d) = 3$, et $\gamma(\sigma_b) = 5$.

■

Nous sommes à présent prêts à décrire notre arbre de génération. La racine de l'arbre est la permutation (1) à 1 élément. Chaque nœud au niveau $n \geq 1$ possède $n + 1$ enfants.

Soient σ une permutation de taille $n - 1$, et $1 \leq i \leq n$. Le i-ème enfant de la permutation σ est la permutation σ', obtenue en ajoutant n en point fixe à σ, puis en appliquant les opérations suivantes, où $\gamma = \gamma(\sigma)$, $p = p(\sigma)$ et $p' = p'(\sigma)$:

3. Le dérangement $\delta(\sigma)$ d'une permutation σ est formé par les non points fixes de σ.
4. La condition de non-spécialité de σ est exactement celle qui fait que $p(\sigma)$ est bien défini.

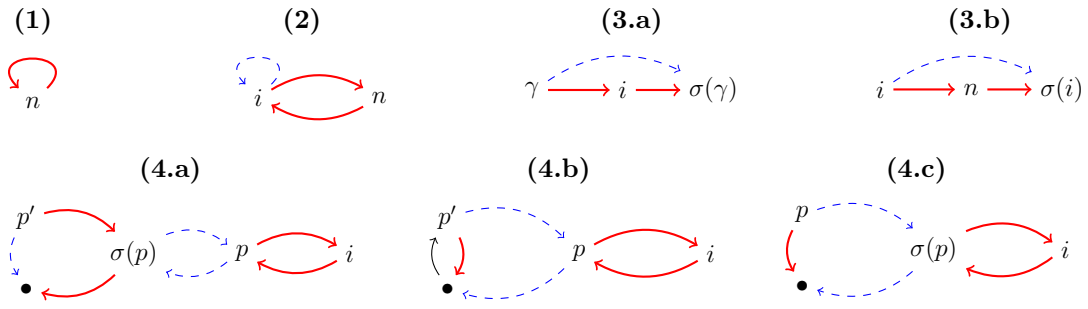


Figure 4.17 – Illustration des différentes règles de l'arbre de génération où $\sigma \in \mathcal{S}_{n-1}$.

1. Si σ est spéciale et $i = n$: aucune autre opération.
2. Si σ est spéciale et $\gamma < i < n$: insérer n après i.
3. Si $i \leq \gamma$:
   (a) Si $i = \sigma(i)$: insérer i après γ.
   (b) Si $i \neq \sigma(i)$: insérer n après i.
4. Si σ est non-spéciale et $i > \gamma$:
   (a) Si $p \in$ 2-cycle : insérer p après i, puis insérer $\sigma(p)$ après p'.
   (b) Si $p \in$ 3-cycle et $\sigma^{-1}(p) = p'$: insérer p après i.
   (c) Sinon : insérer $\sigma(p)$ après i.

Les 7 cas de figure sont présentés figure 4.17, où les liens en pointillés appartenaient à σ, ceux pleins appartiennent à σ'.

Théorème 4.7. *La description ci-dessus définit un arbre de génération.*

Avant de passer à la preuve, vérifions tout d'abord que σ' est bien définie : les règles 1, 2 et 3 couvrent tous les cas de figure pour les permutations spéciales, et les règles 3 et 4 couvrent ceux des permutations non-spéciales ; l'opération insérer b après a est bien définie dés lors que $a \neq b$, ce qui est bien le cas dans les différentes règles, notamment, cas 3.a) : i est point fixe donc $i < \gamma$; cas 4.a) : par définition de p et p', $\sigma(p) \neq p'$.

Nous pouvons aussi vérifier la transmission héréditaire de la propriété d'être non-spéciale. Seules les règles 3 et 4 s'appliquent aux permutations non-spéciales, et les permutations engendrées par ces règles sont elles-mêmes non-spéciales. En effet la règle 3 produit obligatoirement un cycle de longueur au moins 3 (on insère un point fixe après un non-point fixe) ; $p(\sigma') = p(\sigma)$ et $p'(\sigma') = p'(\sigma)$ ne forment toujours pas un cycle de taille 2 après l'application de la règle 4.

Les permutions spéciales de taille $n-1$, quant à elles, engendrent soit des permutation spéciales (leurs $n - \gamma$ derniers enfants via soit la règle 1 en ajoutant un point fixe, soit la règle 2 en supprimant un point fixe et en ajoutant un cycle de taille 2 à la fin de la permutation) soit des non-spéciales (leurs γ premiers enfants via la règle 3).

Preuve (du théorème 4.7) La preuve consiste à inverser la transformation, c'est-à-dire à trouver un antécédent $(\sigma, i) \in \mathcal{S}_{n-1} \times [\![n]\!]$ pour chaque permutation $\sigma' \in \mathcal{S}_n$, de sorte que σ' est le i-ème enfant de σ dans l'arbre. Notons que cela est suffisant car les ensembles de départ et d'arrivée sont de même cardinalité.

Pour cela, nous allons désigner i et décrire les modifications à effectuer sur σ' pour obtenir σ (les images des éléments non décrits sont identiques entre σ' et σ).

Si σ' est spéciale, on observe deux cas possibles :

1. Si $\sigma'(n) = n$, $\sigma = \sigma|_{[\![n-1]\!]}$ et $i = n$. La règle 1 engendre bien σ' car σ est spéciale aussi.
2. Si $\sigma'(n) \neq n$, n appartient alors à un 2-cycle $(i\ n)$ et on remplace ce cycle par (i) pour obtenir σ. La permutation obtenue est toujours spéciale et on a bien $i = \sigma'(n) > \gamma(\sigma)$ comme σ' est spéciale, donc la règle 2 s'y applique et σ' est bien le i-ème enfant de σ.

Si σ' n'est pas spéciale, il existe 2 cas suivant la taille du cycle de $\gamma(\sigma')$:

3. Si $\gamma(\sigma')$ est dans un cycle de taille au moins 3, on a deux sous-cas :
   (a) Si $\sigma'(n) = n$, σ est construite en retirant $i = \sigma'(\gamma(\sigma'))$ de son cycle et l'ajoutant en point fixe à σ ; comme $\gamma(\sigma) = \gamma(\sigma')$, la règle 3.($a$) appliquée à σ produit bien σ' comme $i < \gamma(\sigma)$.
   (b) Si $\sigma'(n) \neq n$, alors $\gamma(\sigma') = n$, et on obtient σ en retirant n de son cycle et $i = \sigma'^{-1}(n)$. Comme le cycle de i est de taille au moins 2 dans σ, la règle 3.(b) produit bien σ'.
4. Si $\gamma(\sigma')$ est dans un cycle de taille 2, on a $i = \gamma(\sigma')$ et il y a 3 sous-cas où i est point fixe dans σ (dans chacun d'entre eux $\gamma(\sigma) < i$ car les points fixes ne sont pas modifiés) :
   (a) Si $\sigma'(i) = p(\sigma')$ et $p'(\sigma')$ est dans un cycle de taille au moins 3 : on obtient σ en retirant $x = \sigma'(p'(\sigma'))$ de son cycle et en ajoutant le 2-cycle $(x\ p(\sigma'))$ à σ. Les éléments $p(\sigma')$ et $p'(\sigma')$ ne sont toujours pas dans le même cycle dans σ, donc $p(\sigma) = p(\sigma')$ et $p'(\sigma) = p'(\sigma')$ et σ est donc spéciale. La règle 4.(a) appliquée à (σ,i) produit bien σ'.
   (b) Si $\sigma'(i) = p(\sigma')$ et $p'(\sigma')$ est dans un cycle de taille 2 : on produit σ en formant le cycle de longueur 3 : $(p'\ p\ \sigma'(p'))$, où $p = p(\sigma') = p(\sigma)$ et $p' = p'(\sigma') = p'(\sigma)$ car les deux éléments ne changent pas, étant dans un 3-cycle. La règle 4.(b) produit effectivement σ'.
   (c) Sinon, c'est-à-dire $\sigma'(i) \neq p(\sigma')$: on insère $\sigma'(i)$ après $p(\sigma')$ afin d'obtenir σ ; $p(\sigma')$ est forcément dans un cycle de taille au moins 3 ; p et p' ne changent pas entre σ' et σ, comme précédemment. De plus, soit p se retrouve dans un cycle de taille au moins 4 dans σ, ou p est dans un 3-cycle, mais dans ce cas $\sigma^{-1}(p) \neq p'$ car sinon ils auraient formé un 2-cycle dans σ', contredisant leur définition. Dans les deux cas, la règle 4.(c) s'applique à (σ,i) et produit σ'.

□

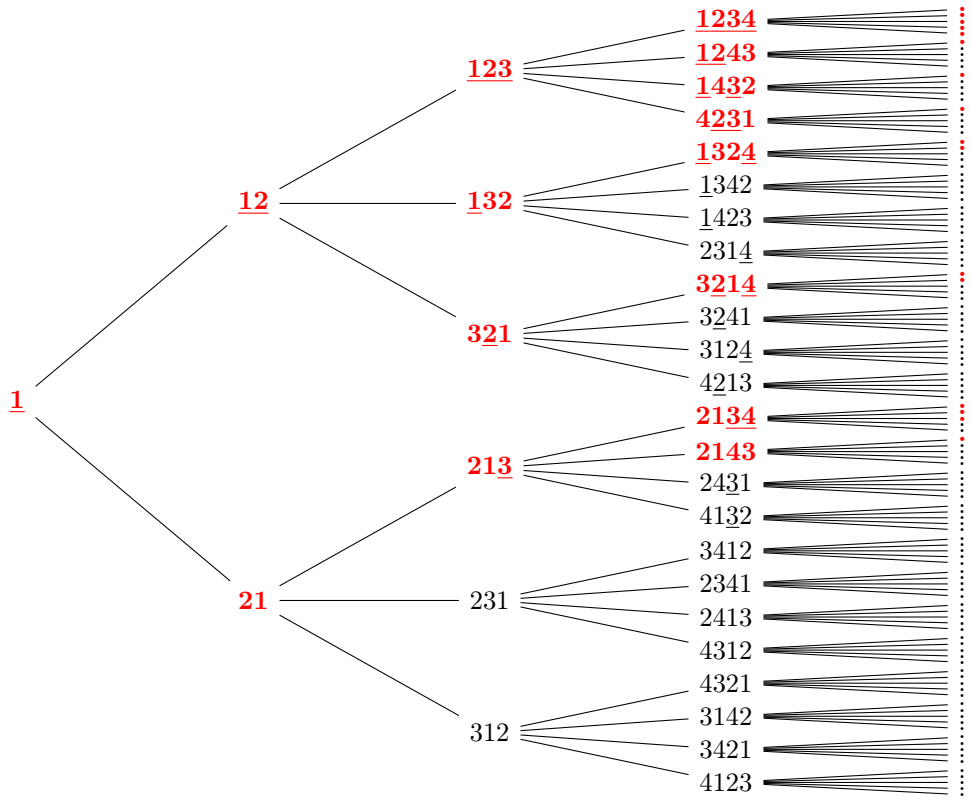


Figure 4.18 – Les cinq premiers niveaux de notre arbre de génération.

Nous terminons cette section en montrant comment notre arbre peut être implémenté de sorte que la n-ième permutation obtenue par une descente arbitraire dans l'arbre soit calculée en temps $\mathcal{O}(n)$. Pour encoder une descente jusqu'au niveau n, nous pouvons par exemple utiliser un code de Lehmer : une séquence de n nombres telle que le i-ème nombre est compris entre 1 et i.

Proposition 4.8. *Calculer la permutation de taille n correspondant à un code de Lehmer de longueur n prend un temps $\mathcal{O}(n)$.*

Preuve La construction décrite est entièrement réalisable à partir de l'opération *insérer après*. De plus, comme l'élément inséré est toujours un point fixe (règle 1, 2 et 3), ou bien la transformation travaille uniquement sur les images de $p(\sigma)$ et $p'(\sigma)$ (règle 4), il est clair que la construction peut être appliquée en temps constant avec la connaissance de $p(\sigma)$ et $p'(\sigma)$ en représentant σ comme un tableau d'entiers.

À partir du moment où la permutation σ devient non-spéciale, $p(\sigma)$ et $p'(\sigma)$ ne font plus que décroître (via la règle 3 de la description de notre arbre de génération) ; ceci nous garantit que la construction d'une permutation de taille n selon notre arbre de génération prend un temps $\mathcal{O}(n)$: chaque descente dans l'arbre sans le calcul de p et p' coûte $\mathcal{O}(1)$, et la mise à jour de p et p' coûte $\mathcal{O}(n)$ au total. □

Remarque La proposition précédente nous indique qu'une permutation aléatoire uniforme de taille n peut être générée en temps $\mathcal{O}(n)$ en supposant que le générateur d'aléa prend un temps constant.

Optimalité

Dans notre arbre de génération, lorsque la règle 1 est utilisée, $\mathrm{fp}(\sigma') = \mathrm{fp}(\sigma) + 1$ alors que pour la règle 2, $\mathrm{fp}(\sigma) = \mathrm{fp}(\sigma) - 1$; dans les autres cas le nombre de points fixes ne changent pas. Ceci permet d'affirmer que seules les 2^{n-1} permutations spéciales, parmi les $n!$ du niveau n, sont *changeantes* dans l'arbre au niveau n. Ce sont donc les seules permutations à avoir un nombre de points fixes différent de leur père. Ce nombre est à comparer avec les $2(n-1)!$ permutations changeantes de l'arbre par insertion dans les cycles, et les $\geq (n-1)!$ de celui par décalage.

Ce nombre de permutations changeantes est optimal pour tout arbre de génération.

Proposition 4.9. *Dans tout arbre de génération uniforme pour les permutations, le nombre de permutations au niveau n ayant un père avec un nombre de points fixes différent, est au minimum de* 2^{n-1}.

Preuve Soit d_n^k le nombre de (n,k)-permutations, *i.e.* permutations de taille n ayant k points fixes. Ces nombres sont communément appelés *nombres de rencontres* (voir *rencontres numbers*, séquence OEIS/A000166 [87]), et sont liés aux nombres de dérangements par $d_n^k = \binom{n}{k} d_{n-k}$ (une (n,k)-permutation est formée d'un ensemble de k points fixes et d'un dérangement des $n-k$ non-points fixes).

D'après la récurrence sur le nombre de dérangements $d_n = nd_{n-1} + (-1)^n$, on obtient directement $d_n^k = n \cdot d_{n-1}^k + \binom{n}{k} \cdot (-1)^{n-k}$.

Lorsque $n-k$ est impair, $d_n^k < n \cdot d_{n-1}^k$, donc les $(n-1,k)$-permutations possèdent assez d'enfants dans un arbre de génération pour couvrir toutes les (n,k)-permutations. *A contrario*, lorsque $n-k$ est pair, $d_n^k > nd_{n-1}^k$, donc au minimum $\binom{n}{k}$ des (n,k)-permutations doivent posséder un père dans l'arbre qui ne soit pas une $(n-1,k)$-permutation. Le nombre de permutations de taille n qui n'ont pas le même nombre de points fixes que leur père est donc au moins

$$\sum_{\substack{k=0 \\ n-k \text{ pair}}}^{n} \binom{n}{k} = \sum_{\substack{k=0 \\ k \text{ pair}}}^{n} \binom{n}{k} = 2^{n-1}$$

où nous retrouvons le nombre de sous-ensembles pairs de $[\![n]\!]$.

□

4.3.4 Génération uniforme de dérangements

Nous présentons une première application probabiliste de notre arbre de génération : la génération uniforme de dérangements.

La propriété héréditaire de préservation du nombre de points fixes des permutations non-spéciales à leurs descendants, peut être exploitée afin de concevoir des algorithmes de génération aléatoire parcimonieux avec leur source d'aléa.

Une instance d'un tel algorithme est la tâche d'échantillonner uniformément un dérangement de manière efficace, à la fois en complexité temps/espace et en complexité d'aléa. Le coût aléatoire de l'algorithme peut être estimé de deux manières différentes, que nous analysons toutes deux : le nombre de bits aléatoires utilisés en moyenne, et le nombre d'appels à un générateur d'entiers $\texttt{Random}(k)$ de coût unitaire, lequel retourne un entier uniforme sur $[\![k]\!]$.

Un algorithme simple de rejet permet de générer un dérangement uniforme de taille n. Pour cela, il suffit de générer des permutations uniformes de taille n, jusqu'à en obtenir une ne possédant pas de points fixes. Comme $d_n = \lceil n!/e \rfloor$, cette méthode simple de rejet n'a besoin de générer que, en moyenne, $e + \mathcal{O}(1/n!)$ permutations de taille n. En outre, cet algorithme utilise en moyenne $en + \mathcal{O}(1/(n-1)!)$ tirages aléatoires en étant de complexité moyenne linéaire, dès lors qu'une méthode standard de génération de permutations est utilisée, telle que le mélange de Fisher-Yates aussi appelé *Knuth shuffle* (voir [49, 38, 66]). Sachant que seuls les dérangements seront retenus à la fin, la méthode peut être améliorée en utilisant un mécanisme de *rejet anticipé*, qui consiste à redémarrer la procédure à partir du moment où le résultat final est sûr de contenir au moins un point fixe. Des calculs approximatifs nous montrent que le nombre de tirages aléatoires est de $(e-1)n+o(n)$ en moyenne dans ce cas.

Dans la littérature, Martinez *et al.* [72] propose un équivalent du mélange de Fisher-Yates pour les dérangements, donnant un algorithme de complexité temps bornée [5] qui utilise en moyenne $2n+o(n)$ appels à un générateur aléatoire. Dans [4], une étude expérimentale est menée afin de comparer cette méthode avec la méthode de génération par rejet anticipé, qui aboutit à des performances comparables en pratique ; une extension de l'algorithme de Martinez *et al.* aux permutations ne possédant que des cycles de longueur au moins $m \geq 3$ est aussi donnée.

Dans [72], les nombres d_n sont utilisés par l'algorithme dans le but de conserver la distribution uniforme lors de la génération de dérangements. Cette idée peut être réutilisée avec la bijection τ_n de Rakotondraja [84] sur les dérangements afin d'obtenir un autre algorithme de génération. Partant d'un dérangement uniforme δ de taille $n-1$ impaire, nous fabriquons un dérangement δ' de taille $n+1$ de la manière suivante :

— Avec probabilité n/d_{n+1}, nous générons un entier i uniforme sur $[\![n]\!]$ et obtenons $\delta' = \tau_{n+1}(\Delta_n, i)$, où $\Delta_n = (1\ 2)\dots(n-1\ \ n)$ est le dérangement critique de taille n.

— Avec probabilité complémentaire $1 - n/d_{n+1}$, nous générons deux uniformes i,j, respectivement sur $[\![n]\!]$ et $[\![n+1]\!]$, puis obtenons le dérangement suivant $\delta' = \tau_{n+1}(\tau_n(\delta,i), j)$.

5. Rappel : on suppose ici que le générateur d'aléa prend un temps constant.

Si un dérangement de taille n paire est visé, le dérangement critique Δ_n est renvoyé avec probabilité $1/d_n$, sinon un dérangement uniforme δ de $\mathcal{D}_{n-1}$ est construit via la méthode précédente ; il suffira alors d'appliquer dans ce dernier cas $\tau_n(\delta,i)$ où i est uniforme sur $[\![n]\!]$.

Cet algorithme a une complexité en temps linéaire dans le pire des cas [6] et nécessite d'appeler au maximum $(3/2)n$ fois la fonction `Random()` (augmenter la taille de 2 unités correspond au maximum à 3 appels).

Nous décrivons ci-après un algorithme par rejet, basé sur notre arbre de génération, qui nécessite en moyenne uniquement $n+\mathcal{O}(1)$ appels à un générateur aléatoire, et de complexité en temps linéaire en moyenne. Cette complexité est similaire à l'algorithme esquissé à la fin de la note [32], se basant sur un encodage par code de Lehmer et une bijection spécifique sur les permutations.

Nous proposons l'algorithme suivant pour générer un dérangement uniforme de taille n :

Algorithme 21 DérangementUniforme(n)

1. Descendre aléatoirement dans l'arbre jusqu'à obtenir une permutation σ soit non-spéciale, soit de taille n, suivant la situation qui arrive en premier.
2. Si la permutation obtenue σ est un dérangement, continuer la descente jusqu'au niveau n et retourner la permutation finale obtenue ; sinon répéter l'étape 1.

Proposition 4.10. DérangementUniforme*(n) retourne un dérangement uniforme de taille n.*

Preuve L'algorithme suit un schéma de rejet anticipé : une permutation uniforme de taille n est construite, mais dès lors que la permutation générée est sûre de contenir des points fixes, l'algorithme redémarre son processus de génération.

Lorsque la génération aboutit, le résultat est bien un dérangement uniforme de taille n.

□

Proposition 4.11. *L'algorithme proposé utilise en moyenne $n + \mathcal{O}(1)$ appels au générateur `Random()` lors de la génération d'un dérangement de taille n.*

Preuve Soit $A = A(n)$ la variable aléatoire comptant le nombre d'appels à `Random()` utilisé par l'algorithme avec comme entrée n. Nous séparons le coût en deux contributions correspondant aux deux phases de l'algorithme, de telle sorte que $A = C_1 + C_2$,

6. Plus précisément, la bijection d'origine de [84] s'implémente facilement en temps linéaire lorsque les antécédents sont stockés en même temps que le dérangement. La bijection peut aussi être modifiée (la règle 4 de notre arbre de génération définit une telle bijection) afin de n'avoir besoin que des images du dérangement stocké dans un tableau.

où C_1 compte les appels effectués lors de la première phase, *i.e.* le nombre d'appels au générateur aléatoire (incluant les tentatives se terminant par un échec) jusqu'à l'obtention d'un dérangement soit non-spéciale ou de taille n ; C_2 compte quant à elle seulement les appels effectués lors de la phase 2 de l'algorithme.

Nous avons $\mathbb{E}(C_1) = \mathbb{E}(C)/p$, où C est le nombre d'appels effectués lors d'une tentative de l'étape 1, et p la probabilité de succès de chaque tentative. Nous savons déjà que $pd_n/n! \geq 1/e - 1/n!$, d'où $1/p = e + \mathcal{O}(1/n!)$.

Nous calculons à présent une borne supérieure sur $\mathbb{E}(C)$. Pour que m appels (avec $m \geq 1$) soient effectués avant un rejet lors d'une même tentative, la permutation de taille m courante doit forcément être spéciale, ce qui se produit avec probabilité $2^{m-1}/m!$, donc $\mathbb{P}(C \geq m) = 2^{m-1}/m!$.

En sommant sur les valeurs possibles de m, nous obtenons

$$\mathbb{E}(C) = \sum_{m=1}^{n-1} \mathbb{P}(C \geq m) \leq \sum_{m=1}^{\infty} \frac{2^{m-1}}{m!} = -\frac{1}{2} + \frac{1}{2} \sum_{m=0}^{\infty} \frac{2^m}{m!} = \frac{e^2 - 1}{2}.$$

De plus, C_2 est borné par $n - 3$, car les plus petites permutations non-spéciales sont de taille 3, conduisant à une espérance elle-même bornée par $n - 3$. Par linéarité de l'espérance, nous obtenons que le nombre moyen total d'appels utilisé dans l'algorithme est

$$\mathbb{E}(A) = \mathbb{E}(C_1) + \mathbb{E}(C_2) \leq n - 3 + e(e^2 - 1)/2 + \mathcal{O}(1/n!).$$

□

Afin d'estimer maintenant le nombre de bits aléatoires utilisés au cours de l'algorithme, nous supposons que la primitive `Random(k)` est optimale dans le sens décrit par [68]. Ceci implique que, pour tout $k > 2$, le nombre moyen de bits utilisés par `Random(k)` est borné par $2+\log_2(k)$. Cette borne est au maximum de $2+\log_2(n)$, car k n'est jamais supérieur à n. Finalement, en multipliant simplement par l'espérance du nombre d'appels à `Random(k)`, nous déduisons la proposition suivante.

Proposition 4.12. DérangementUniforme*(n) utilise* $n \log_2(n) + o(n \log(n))$ *bits uniformes en moyenne.*

Nous n'avons pas essayé d'obtenir une borne plus précise, car la même borne inférieure est aussi valide : le nombre moyen de bits uniformes utilisés ne peut être inférieur à $\log_2(d_n) = n \log_2(n) + \mathcal{O}(n)$.

Remarque L'algorithme présenté peut être amélioré en rejetant (*i.e.* en recommençant à zéro) aussi lorsque la permutation courante contient plus de points fixes que le nombre de tours restants pour atteindre n. En effet, dans ce cas de figure, la permutation finale ne peut être un dérangement car au maximum le nombre de points fixes diminue de 1 par niveau dans l'arbre. Néanmoins, l'analyse ci-avant montre que cette possibilité n'apparaît que très rarement lorsque n dépasse quelques unités, donc une telle amélioration ne changerait pas la complexité globale de l'algorithme.

4.3.5 Simulation de la loi de Poisson de paramètre 1

Nous présentons dans cette section une seconde application probabiliste de notre arbre de génération : la génération de variables de Poisson. Plus précisément, notre arbre de génération peut être utilisé afin de générer de façon combinatoire une variable aléatoire distribuée selon la loi de Poisson de paramètre 1, où chaque entier $k \geq 0$ est obtenu avec probabilité $1/(ek!)$.

Plusieurs méthodes de génération existent d'ores et déjà afin de simuler la distribution de Poisson (voir [33] pour la plupart d'entre elles). Ces algorithmes sont pour la plupart efficaces mais manipulent des nombres rationnels et irrationnels. Au contraire, notre algorithme est purement combinatoire par nature et ne travaille qu'avec de petits entiers.

L'algorithme se décrit en une ligne : descendre aléatoirement dans l'arbre jusqu'à obtenir une permutation non-spéciale, puis renvoyer son nombre de points fixes.

L'analyse effectuée dans la preuve de la proposition 4.11 nous indique qu'un tel algorithme ne nécessitera en moyenne que $\frac{e^2-1}{2} \approx 3.2$ appels à un générateur de nombres aléatoires.

Nous rappelons qu'une *descente aléatoire* consiste à choisir une branche uniformément parmi les enfants du nœud courant, puis *se déplacer* sur l'enfant atteint en suivant la branche ainsi choisie. Comme dans la description de notre arbre de génération § 4.3.3, les permutations ne restent spéciales que lorsque les règles 1 et 2 sont utilisées, nous n'avons à retenir la valeur que de 3 paramètres : la taille de la permutation courante, le plus grand non-point fixe de la permutation courante et le nombre de points fixes qui sera renvoyé à la fin ; respectivement n, g et k dans l'algorithme 22.

Algorithme 22 POISSON1

$n \leftarrow 1$, $g \leftarrow 0$, $k \leftarrow 1$
répéter
  $i \leftarrow Random(n+1)$
  si $i = n+1$ **alors**
    $k \leftarrow k+1$
  sinon si $i > g$ **alors**
    $k \leftarrow k-1, g \leftarrow n+1$
  sinon
    retourner k
  $n \leftarrow n+1$

Avant de passer à la preuve que l'algorithme présenté simule bien la loi Poi(1), nous formulons la remarque suivante afin de calculer la probabilité de retourner k au tour n.

Remarque Parmi les enfants des permutations spéciales de taille n, ceux qui sont non-spéciaux sont obligatoirement issus de la règle 3.

Parmi ceux-ci, $(n-k)\binom{n+1}{k}$ ont k points fixes, un dénombrement qu'on retrouve soit en sommant sur γ : $\sum_{\gamma=n-k}^{n}\binom{\gamma-1}{\gamma+k-n}\gamma = (n-k)\binom{n+1}{k}$, soit en remarquant que lorsque l'on fixe l'ensemble de leurs points fixes S et l'entier $i \in [\![n]\!] \setminus S$ utilisé dans les règles 3.(a) et 3.(b), le reste de la permutation s'en déduit.

Proposition 4.13. *L'entier retourné par l'algorithme* Poisson1 *est distribué selon la distribution de Poisson de paramètre* 1.

Preuve Nous remarquons dans un premier temps que l'algorithme présenté implémente effectivement le sous-arbre des permutations spéciales de notre arbre de génération, en ne conservant que la taille, le plus grand point fixe et le nombre de points fixes de la permutation courante.

Nous montrons que la probabilité que l'algorithme retourne k est $1/(k!e)$.

Remarquons tout d'abord que le tour n est atteint avec probabilité $2^{n-1}/n!$ (avec probabilité complémentaire une permutation non-spéciale a été produite au cours d'un tour antérieur). Sachant que le tour n est atteint, à la fin du tour, la *permutation courante* est uniforme sur les enfants des permutations spéciales de taille n. Comme parmi ces permutations de taille $n+1$, $(n-k)\binom{n+1}{k}$ ont k points fixes, la probabilité de s'arrêter à la fin du tour n en renvoyant k, est égale à la probabilité d'atteindre le tour n multipliée par $\frac{(n-k)\binom{n+1}{k}}{(n+1)2^{n-1}} = \frac{(n-k)n!}{2^{n-1}k!(n+1-k)!}$.

En sommant sur tous les tours n, où k peut être renvoyé (uniquement si $n \geq k$ et $n-k$ est pair), nous trouvons

$$\begin{aligned}\mathbb{P}(\text{Poisson1}() = k) &= \sum_{\substack{n=k \\ n-k \text{ pair}}}^{\infty} \frac{2^{n-1}}{n!}\frac{(n-k)n!}{2^{n-1}k!(n+1-k)!} = \frac{1}{k!}\sum_{\substack{n=k \\ n-k \text{ pair}}}^{\infty}\frac{n-k}{(n+1-k)!} \\ &= \frac{1}{k!}\sum_{\substack{j=0 \\ j \text{ pair}}}^{\infty}\frac{1}{j!} - \frac{1}{(j+1)!} = \frac{1}{k!}\sum_{j=0}^{\infty}\frac{(-1)^j}{j!} = \frac{1}{k!e}\end{aligned}$$

où $n-k$ a été remplacé par j à la seconde ligne d'équation.

□

Remarque Les propriétés de notre arbre de génération nous garantissent que le nombre de points fixes d'une permutation issue d'une descente aléatoire converge presque sûrement, et nous avons déjà remarqué que l'algorithme présenté retourne précisément la limite du nombre de points fixes de cette descente. Or, comme nous l'avons déjà mentionné dans l'introduction de cette section, il est connu que cette limite est distribuée comme Poi(1).

Notre preuve permet de redémontrer ce résultat, qui est un des premiers théorèmes prouvés en théorie des probabilités par de Montfort en 1714 [31].

Nous analysons maintenant le coût en terme de bits aléatoires uniformes de l'algorithme présenté.

Proposition 4.14. *Notre algorithme* POISSON1 *utilise en moyenne entre* 6.89 *et* 6.9 *bits uniformes, lorsque la primitive* `Random` *est optimale en terme de bits utilisés.*

Preuve Soit C le nombre de bits aléatoires utilisés par l'algorithme et C_n de bits consommés au tour n (potentiellement 0 si l'algorithme s'arrête avant le tour n), de telle sorte que $C = \sum_{n=1}^{\infty} C_n$.

Nous avons $\mathbb{E}(C_n) = \frac{2^{n-1}}{n!}\mathbb{E}(U_{n+1})$, où U_n est le nombre de bits utilisés par un générateur d'uniformes sur $[\![n]\!]$ comme décrit dans [68], soit `Random`$(n+1)$ ligne 3 de l'algorithme ; le facteur $2^{n-1}/n!$ est simplement la probabilité d'atteindre le niveau n de l'arbre, *i.e.* la permutation courante est spéciale. L'inégalité $\log_2(n) \leq \mathbb{E}(U_n) \leq \log_2(n) + 2$ est valide pour tout n, l'utiliser telle quelle nous donne une borne supérieure d'environ 11,61 bits uniformes, ne correspondant pas à la borne que nous visons.

À la place, nous calculons exactement $D_k = \sum_{n=1}^{k} \frac{2^{n-1}}{n!}\mathbb{E}(U_{n+1})$ pour une certaine constante k. Les bornes inférieures et supérieures pour $\mathbb{E}(C)$ deviennent donc respectivement

$$D_k + \sum_{n>k} \frac{2^{n-1}\log_2(n+1)}{n!} \quad \text{et} \quad D_k + \sum_{n>k} \frac{2^{n-1}(2+\log_2(n+1))}{n!}.$$

Pour n fixé, $\mathbb{E}(U_n)$ peut être simplement obtenu en considérant le développement binaire de $1/n$, comme proposé dans [68]. Pour obtenir les estimations énoncées dans la proposition, nous avons calculé les bornes explicitement pour $k = 10$. Les valeurs de $\mathbb{E}(U_n)$ pour n de 2 à 10 sont : 1, 8/3, 2, 18/5, 11/3 24/7, 3, 14/3, 23/5, 150/31, ce qui donne $D_{10} = 3031201/439425 \approx 6{,}898$, qui est de plus une borne inférieure à l'espérance de la complexité en bits aléatoires.

Pour la borne supérieure, nous majorons la quantité suivante

$$\sum_{n>10} \frac{2^{n-1}(2+\log_2(n+1))}{n!} \leq \sum_{n\geq 10} \frac{2^n}{n!} = e^2 - \sum_{n=0}^{9} \frac{2^n}{n!} = e^2 - \frac{2977}{405} \simeq 0{,}00034$$

où nous avons utilisé l'inégalité $2 + \log_2(n+1) \leq n$ (qui est valide pour $n \geq 11$).

Nous avons en effet un coût moyen, en nombre de bits aléatoires utilisés, compris entre 6,89 et 6,9.

□

Notre algorithme POISSON1 de simulation de la loi de Poisson, bien qu'il soit efficace, n'est pas optimal ; encore une fois en s'appuyant sur les résultats de [68], un algorithme optimal devrait utiliser, en moyenne, moins de $2 + E$ bits, où $E = 1 + e^{-1}\sum_{k\geq 0} \log_2(k!)/k! \simeq 1{,}89$ est l'entropie binaire de la distribution de Poisson ; on notera toutefois, qu'un tel algorithme aura besoin du développement binaire de toutes les probabilités individuelles de la distribution Poi(1), $p_k = 1/(k!e)$.

En développant les premiers chiffres binaires des probabilités de la loi de Poisson (jusqu'à 2^{-25}, soit p_{10}), nous obtenons que l'algorithme optimal utilise au minimum 3,36 bits uniformes.

Il est facile de modifier l'algorithme afin d'utiliser moins de bits uniformes en moyenne tout en restant de complexité constante, mais au détriment de perdre sa simplicité.

Premièrement, nous n'avons pas besoin de la valeur précise de i dans l'algorithme, seulement de savoir si $i < g$, $g < i < n+1$ ou $i = n+1$. Cette loi ternaire [7] $p(g,n)$ peut donc être simulée en moins de $\log_2(3) + 2$ bits en moyenne. Néanmoins, cette borne n'améliore pas l'encadrement de la complexité en bits de notre algorithme.

En revanche, si on remplace le générateur `Random(n)` par un générateur optimal pour la distribution $p(g,n)$, la complexité en bits est nettement améliorée. Un tel générateur peut être implémenté en arrêtant la génération à partir du moment où la valeur de i est sûre soit d'être strictement inférieure à g, soit comprise entre g et $n+1$. Nous pouvons calculer une borne supérieure en utilisant un tel générateur sur les premiers niveaux de l'arbre, puis en utilisant notre borne supérieure déjà calculée pour les niveaux suivants ; en utilisant les 5 premiers niveaux, cette borne est de 4,69 bits en moyenne.

Une dernière méthode changeant profondément l'algorithme, est de simuler optimalement lors d'une étape de pré-calcul le résultat du n-ème niveau de l'arbre (en temps $\mathcal{O}(n!)$) afin de calculer une table de taille $\mathcal{O}(n^2)$ contenant les probabilités de retourner directement les valeurs 0 à $n-3$ et les probabilités de démarrer l'algorithme Poisson1 en initialisant les variables k et g par un des $\mathcal{O}(n^2)$ couples possibles. Comme n est fixe lors de la simulation ultérieure de variables selon la loi de Poisson, la complexité d'un tel algorithme reste en $\mathcal{O}(1)$.

Par exemple, lorsque $n = 5$, il existe 10 possibilités suivant le nombre de permutations de chaque catégorie, qui nous fournit l'algorithme suivant :

- avec probabilité 44/120 retourner 0 (dérangements non-spéciaux)
- avec probabilité 40/120 retourner 1 (perm. non-spéciales à 1 point fixe)
- avec probabilité 20/120 retourner 2 (perm. non-spéciales à 2 points fixes)
- dans les autres cas démarrer l'algorithme Poisson1 avec (en comptant le nombre de $(5,k)$-permutations dont le plus grand non-point fixe est g) :
  - $k = 5$, $g = 0$ avec probabilité 1/120
  - $k = 3$, $g = 5$ avec probabilité 4/120
  - $k = 3$, $g = 4$ avec probabilité 3/120
  - $k = 3$, $g = 3$ avec probabilité 2/120
  - $k = 3$, $g = 2$ avec probabilité 1/120
  - $k = 1$, $g = 5$ avec probabilité 4/120
  - $k = 1$, $g = 4$ avec probabilité 1/120

qui utilise moins de 3,66 bits en moyenne (cette borne est obtenue en calculant le nombre de bits optimal pour la distribution décrite, et en y ajoutant la borne supérieure calculée dans la preuve précédente).

Ce calcul nous fournit aussi une borne supérieure pour l'algorithme optimal, qui pour $n = 5$, améliore déjà la borne générale de l'entropie plus deux.

7. Ternaire dans le sens où son support est de taille 3 pour les trois événements $i < g$, $g < i < n+1$ et $i = n+1$.

4.4 Simulation efficace de la loi Bin(n, k/n)

Nous présentons dans cette section comment l'arbre de génération que nous avons décrit va permettre d'améliorer l'algorithme du premier doublon, présenté à la section 4.2. Cette amélioration permettra de passer d'un coût moyen de l'ordre de $\sqrt{n}$ à $\mathcal{O}(1)$ (voir proposition 4.6, 116).

Le paragraphe qui suit montre de plus comment l'algorithme peut être adapté pour obtenir aussi une variable distribuée comme $\mathrm{Bin}(n, k/n)$.

4.4.1 Algorithme du premier doublon amélioré

Nous rappelons que le but de l'algorithme que nous allons présenter est de simuler la distribution $\mathrm{Bin}(n, 1/n)$ à partir de tirages aléatoires uniformes sur un ensemble de taille n, sans connaître la valeur de n. Pour rappel, l'algorithme du premier doublon (cf. algorithme 20, page 115) consiste à générer une permutation σ_M uniforme de taille M où M est une variable aléatoire comptant le nombre d'uniformes sur V (l'ensemble sous-jacent de taille n) tirées avant d'obtenir un doublon, *i.e.* un certain élément $x \in V$ pour la seconde fois. Une fois la permutation σ_M générée, son nombre de points fixes est renvoyé. Si la taille de V est n, alors $\mathrm{fp}(\sigma_M) \sim \mathrm{Bin}(n, 1/n)$.

L'amélioration que nous proposons consiste à simuler *par anticipation* directement $\mathrm{fp}(\sigma_M)$ avant d'obtenir le premier doublon et donc de connaître M, qui arrive aux environs de $1.25\sqrt{n}$ tirages en moyenne. L'idée est la suivante : nous allons générer la permutation uniforme σ_M en descendant dans notre arbre de génération, mais en abrégeant le processus dès lors que $\mathrm{fp}(\sigma_M)$ est acquis. La descente dans l'arbre est pour cela couplée à des tirages sur V.

Plus précisément, nous générons σ_M en partant de σ_1 : la permutation d'un unique élément de V (correspondant au premier tirage sur V). Puis, nous générons σ_{i+1} à partir de σ_i en descendant aléatoirement d'un niveau dans notre arbre de génération (σ_{i+1} est une permutation des $i+1$ premiers tirages sur V, tous distincts). Lorsqu'un doublon est obtenu, le processus s'arrête et le nombre de points fixes de σ_M est renvoyé. Cet algorithme correspond juste à la mise en parallèle de la génération de la permutation finale σ_M et des tirages sur V.

Nous pouvons à présent utiliser la propriété essentielle de notre arbre de génération : lors du processus de simulation, le nombre de points fixes de la permutation finale σ_M est fixé à partir du tour $i < M$ lorsque σ_i est une permutation non-spéciale, ou au tour M si σ_M est spéciale. Lorsque $\mathrm{fp}(\sigma_M)$ est définitivement acquis, l'algorithme peut se terminer sans continuer la descente jusqu'au niveau M.

Comme dans le cas de la simulation de la loi de Poisson, seul le sous-arbre des permutations spéciales est utilisé. Nous pouvons donc obtenir la procédure décrite ci-avant en adaptant l'algorithme de simulation de loi Poi(1) afin d'intégrer des tirages d'uniformes sur V. L'algorithme 23 SIM-DOUBLON que nous présentons intègre une légère amélioration supplémentaire (économisant environ un tirage) : la descente dans l'arbre est effectuée avant le tirage d'uniforme sur V.

Algorithme 23 Sim-Doublon

```
1: m ← 1, g ← 0, j ← 1
2: S ← {Uniforme(V)}
3: répéter
4:     i ← Random(m + 1)
5:     si i = m + 1 alors
6:         j ← j + 1
7:     sinon si i > g alors
8:         j ← j − 1, g ← m + 1
9:     sinon
10:        retourner j
11:    x ← Uniforme(V)
12:    si x ∈ S alors
13:        si g = m + 1 alors
14:            retourner j + 1
15:        sinon
16:            retourner j − 1
17:    sinon
18:        S ← S + x
19:    m ← m + 1
```

Ainsi, au i-ème tour de boucle, la permutation σ_{i+1} simulée après les lignes 4-10 (dont on ne retient que le nombre de points fixes j et le plus grand non-point fixe g) est une permutation de taille $i+1$. Lorsqu'un doublon est détecté au tour i, alors il faudra remonter au père de σ_{i+1} pour obtenir une permutation uniforme de la même taille que le nombre d'éléments tirés jusqu'au premier doublon.

Proposition 4.15. *L'algorithme* Sim-Doublon *retourne* $0 \leq j \leq n$ *avec probabilité* b_j^n*, lorsque* $n = |V|$.

Preuve Si le m-ème tour de boucle est le premier où le tirage sur V à la ligne 11 produit un doublon, alors d'après le théorème 4.5, le nombre de points fixes d'une permutation uniforme de taille m est distribué selon la loi $\mathrm{Bin}(n, 1/n)$. Nous allons montrer que la sortie de l'algorithme est distribuée identiquement.

Soit M la variable aléatoire indiquant le tour d'apparition du premier doublon lors de tirages uniformes et indépendants sur V, dont les i premiers correspondent aux résultats de la ligne 11 et où i est le tour d'arrêt de l'algorithme.

Étant donné une permutation uniforme σ_M générée à partir des permutations successives $\sigma_1, \ldots, \sigma_{M-1}$ en descendant dans notre arbre de génération, en suivant les choix aléatoires de la ligne 4 lors de l'exécution de l'algorithme sur les i premiers niveaux, nous avons que :

— Soit σ_M est une permutation non-spéciale, alors elle a le même nombre de points fixes que σ_i, avec $i < M$, son dernier ancêtre spécial dans l'arbre ;

dans ce cas, l'algorithme termine forcément par la ligne 10 au i-ème tour de boucle, en renvoyant $\mathrm{fp}(\sigma_i) = \mathrm{fp}(\sigma_M)$.

— Soit σ_M est une permutation spéciale, alors tous ces parents $\sigma_1, \ldots, \sigma_{M-1}$ dans l'arbre sont aussi spéciaux. Par conséquent, l'algorithme atteint obligatoirement le tour M, et deux cas de figure se posent alors :
  — soit σ_{M+1} (la permutation produite au tour M) est une permutation non-spéciale, alors l'algorithme retourne $\mathrm{fp}(\sigma_{M+1}) = \mathrm{fp}(\sigma_M)$ en exécutant la ligne 10 ;
  — soit σ_{M+1} est aussi une permutation spéciale, alors la ligne 11 est exécutée et un doublon est obtenu. Les lignes 12-16 inversent la dernière opération effectuée sur j, de sorte que la valeur retournée soit bien $\mathrm{fp}(\sigma_M)$.

La sortie de l'algorithme suit donc la même distribution que le nombre de points fixes de σ_M.

□

Proposition 4.16. *Quel que soit $|V|$, l'algorithme* SIM-DOUBLON *effectue en espérance moins de $(e^2-1)/2 \leq 3{,}2$ tirages sur V.*

Preuve L'espérance du nombre de tours effectués est

$$\sum_{m=1}^{n} \left(\prod_{i=0}^{m-1} \frac{n-i}{n} \right) \frac{2^{m-1}}{m!}$$

où $\left(\prod_{i=0}^{m-1} \frac{n-i}{n} \right) \frac{2^{m-1}}{m!}$ est la probabilité de faire au moins m tours de boucle : $\prod_{i=0}^{m-1} \frac{n-i}{n}$ étant la probabilité de ne pas avoir tiré un doublon jusqu'au tour m et $\frac{2^{m-1}}{m!}$ est la probabilité que la permutation courante σ_m soit spéciale.

Comme cette somme est bornée par l'espérance du nombre d'appels à *Random*, le calcul déjà effectué dans la preuve de la proposition 4.11 nous permet alors de conclure.

□

4.4.2 Extension au cas $k \geq 2$

Nous présentons comment l'algorithme du premier doublon peut être utilisé afin de simuler la loi $\mathrm{Bin}(n, k/n)$. Pour arriver à nos fins, nous allons simuler un sous-ensemble binomial (de probabilité) k/n de V, *i.e.* un sous-ensemble W de V où $\mathbb{P}(x \in W) = k/n$ et les évènements $\{x \in W\}_{x \in V}$ sont indépendants. Dans le reste de la section k est constant, et $n \geq k$ est la taille de l'ensemble sous-jacent V.

Lors de la description de la première version de l'algorithme du premier doublon (algorithme 20), nous obtenions directement un ensemble binomial $1/n$: les points fixes de la permutation des éléments tirés jusqu'au premier doublon.

À la fin du précédent algorithme, non seulement une variable aléatoire distribuée comme $\mathrm{Bin}(n, 1/n)$ est générée, mais aussi un certain nombre d'éléments de

l'ensemble V sont appris au cours de la simulation. Ce sous-ensemble de V acquis est forcément de taille au-moins égale à la valeur retournée par l'algorithme ; de plus il est au minimum de taille 1.

Nous pouvons donc aussi aisément construire un ensemble binomial $1/n$ dans ce cas : à la place de retourner j, renvoyer les j premiers éléments appris lors des tirages sur V (forcément distincts).

Nous proposons de construire un sous-ensemble U_i binomial i/n de V à partir d'un sous-ensemble U_{i-1} binomial $(i-1)/n$ de V, un sous-ensemble $W \subset V$ de taille $i-1$ et un élément $u \in V \setminus W$, par la procédure suivante :

Algorithme 24 $\mathcal{P}_i(U_{i-1}, W, u)$

1. Construire le sous-ensemble U en utilisant Algorithme-Premier-Doublon où les appels à Uniforme(V) sont remplacés par des appels à Uniforme($V \setminus W$).
2. Ajouter à U chaque élément de W avec probabilité $1/(n-(i-1))$ indépendamment les uns des autres.
3. Le sous ensemble U_i est enfin obtenu en prenant l'union entre U_{i-1} et U.

La seconde étape est résolue en appelant Uniforme($V \setminus W$) et en testant si on a obtenu u. Le générateur Uniforme($V \setminus W$) est quant à lui simulé à partir de Uniforme(V) par des rejets (soit `RVA` § 1.3.2 dans notre modèle d'algorithmes de mise à jour).

Proposition 4.17. *La procédure $\mathcal{P}_i(U_{i-1}, W, u)$ décrite (algorithme 19) construit un sous-ensemble U_i binomial i/n de V.*

Preuve Soit $x \in V$. Par hypothèse, nous avons $x \in U_{i-1}$ avec probabilité $(i-1)/n$.

Le sous-ensemble construit à l'étape 1 est un sous-ensemble binomial $1/(n-(i-1))$ de $V \setminus W$ par la proposition 4.15. Si chaque élément de W y est ensuite ajouté indépendamment avec probabilité $1/(n-(i-1))$ alors l'ensemble obtenu est un sous-ensemble binomial $1/(n-(i-1))$ de V.

Comme le sous-ensemble U est indépendant de U_{i-1}, nous obtenons :

$$\begin{aligned}\mathbb{P}(x \in U_i) &= \mathbb{P}(x \in U) + \mathbb{P}(x \in U_{i-1}) - \mathbb{P}(x \in U, x \in U_{i-1}) \\ &= \frac{1}{n-(i-1)} + \frac{i-1}{n} - \frac{i-1}{n(n-(i-1))} = \frac{i}{n}\end{aligned}$$

Pour obtenir l'indépendance des évènements $\{x \in U_i\}_{x \in V}$, il suffit de considérer que l'ensemble binomial U (respectivement U_{i-1}) est construit à partir de n Bernoulli i.i.d $(B_x)_{x \in V}$ de paramètre $1/(n-(i-1))$ indiquant si $x \in U$, resp. $(B'_x)_{x \in V}$ de paramètre $(i-1)/n$ indiquant si $x \in U_{i-1}$. En tant que fonction déterministe d'ensembles indépendants, les $\{x \in U_i\}_{x \in V}$ sont indépendants.

□

Lorsque W et u sont connus, $((i-1)+\frac{e^2-1}{2})\frac{n}{n-(i-1)} \leq i+2{,}2+\mathcal{O}(1/n)$ appels en moyenne à un générateur d'uniformes sur V sont nécessaires pour construire l'ensemble U_i. Ce coût découle directement du nombre d'appels nécessaires pour simuler Uniforme$(V \setminus W)$, et du coût de l'algorithme du premier doublon donné à la section précédente (voir proposition 4.16).

En identifiant dans un premier temps k sommets $(u_i)_{i\in[\![k]\!]}$, puis en construisant itérativement les ensembles U_1 (algorithme du premier doublon classique), U_2, …, U_k, nous obtenons un coût total asymptotique inférieur à

$$k+\frac{(e^2-1)k+k(k-1)}{2}=\frac{k(e^2+k)}{2}=\mathcal{O}(k^2)$$

pour construire U_k à partir de zéro (la correction s'obtient en conditionnant sur les k sommets $(u_i)_{i\in[\![k]\!]}$).

Remarque À titre indicatif, la borne donnée $\frac{k(e^2+k)}{2}$ dans le calcul précédent est de $2+e^2 \approx 9{,}39$ pour $k=2$, et elle est d'environ 15,58 pour $k=3$.

Nous terminons cette section en montrant comment un coût linéaire en k peut être obtenu, avec probabilité proche de 1, lorsque n est grand.

Algorithme 25 BIN-K-AMÉLIORÉ

1. Identifier k éléments distincts $(u_i)_{i\in[\![k]\!]}$ de V.
2. Construire les sous-ensembles $(U^i)_{i\in[\![k]\!]}$, où chaque U^i est obtenu à partir de l'algorithme du premier doublon sur $V \setminus \{u_1, \dots u_{i-1}\}$.
3. Calculer $U=\bigcup_{i\in[\![k]\!]} U^i$.
4. Ajouter enfin à U les éléments u_1, u_2, …, u_{k-1}, chacun avec probabilité $\frac{k-i}{n-i}$ et ce de manière indépendante.

La première étape est résolue par des appels successifs à Uniforme(V). L'étape 4 se base sur la simulation de Bernoulli$(\frac{k-i}{n-i})$ indépendantes, où chacune d'entre-elles est simulée en tirant un élément v uniforme sur $V \setminus \{u_1, \dots, u_i\}$, puis en vérifiant si $v \in \{u_{i+1}, \dots, u_k\}$.

Proposition 4.18. *La procédure* BIN-K-AMÉLIORÉ *décrite ci-dessus construit un sous-ensemble binomial k/n de V en moins de $k\frac{e^2+3}{2}-1+\mathcal{O}(1/n)$ tirages sur V en moyenne.*

Preuve La preuve se déroule conditionnellement à u_1, u_2, …, u_k, les sommets obtenus à l'étape 1.

Montrons dans un premier temps que l'ensemble construit suit la même distribution que celui obtenu dans la procédure $\mathcal{P}$ décrite précédemment (*i.e.* itérations de k exécutions de l'algorithme 24).

Les ensembles $(U^i)_{i\in[\![k]\!]}$ correspondent bien aux ensembles obtenus lors de l'étape 1 de $\mathcal{P}$, où $W = \{u_1, \dots, u_{i-1}\}$ lors de l'exécution de l'étape 1 de $\mathcal{P}_i$. Lors du calcul de U_k (étape 3 de $\mathcal{P}$ à la k-ème exécution), un élément v est présent dans U_k s'il est présent dans l'un des ensembles $U^1, \dots, U^k$ calculés aux étapes 1, ou si $v \in \{u_1, \dots, u_{k-1}\}$ a été ajouté à l'une des étapes 2.

À la place de simuler les évènements $\{u_i \in U_j\}_{i\in[\![j-1]\!]}$ individuellement, nous allons simuler directement les évènements $\{u_i \in U_k\}_{i\in[\![k-1]\!]}$, qui sont indépendants. Nous avons pour $i \in [\![k-1]\!]$ et $j \geq i+1$, u_i est ajouté à l'exécution de l'étape 2 de $\mathcal{P}_j$ avec probabilité $1/(n-(j-1))$, et ce de manière indépendante pour chaque procédure $(\mathcal{P}_j)_{j\in[\![2,k]\!]}$. Soit $B_{i,j}$ une variable aléatoire de Bernoulli indiquant si u_i a été ajouté lors de l'exécution de $\mathcal{P}_j$. Par hypothèse les $B_{i,j}$ sont indépendantes, et donc nous avons :

$$\begin{aligned}\mathbb{P}(u_i \in U_k) &= 1 - \mathbb{P}(u_i \notin U_k) = 1 - \mathbb{P}\left(\bigcap_{j\in[\![i+1,k]\!]} B_{i,j} = 0\right) \\ &= 1 - \left(\frac{n-i-1}{n-i} \cdot \frac{n-i-2}{n-i-1} \cdots \frac{n-k}{n-k+1}\right) = \frac{k-i}{n-i}.\end{aligned}$$

Enfin remarquons que l'étape 4 de la procédure ajoute bien chaque élément $v \in \{u_1, \dots, u_{k-1}\}$ avec probabilité $\frac{k-i}{n-i}$.

Concernant le coût en nombre de tirages sur V, il faut en moyenne moins de $k\frac{e^2-1}{2} + \mathcal{O}(1/n)$ tirages pour construire les sous-ensembles $(U^i)_{i\in[\![k]\!]}$ via l'algorithme du premier doublon amélioré (algorithme 23, Sim-Doublon). Il faut de plus $k + \mathcal{O}(1/n)$ tirages en moyenne pour résoudre l'étape 1, et $k-1+\mathcal{O}(1/n)$ pour résoudre l'étape 4.

Au total, moins de $k\frac{e^2+3}{2} - 1 + \mathcal{O}(1/n)$ tirages en moyenne sont nécessaires.

□

Au cours de ce chapitre, nous avons vu différents algorithmes de simulation pour la loi $\mathrm{Bin}(n, 1/n)$ n'utilisant jamais n, mais ayant accès à la place à un générateur d'uniformes sur un ensemble de taille n. La complexité de ces algorithmes (en nombre d'appels au générateur et en pas de calculs) est passée au fur et à mesure du chapitre de $\mathcal{O}(n)$, à $\mathcal{O}(\sqrt{n})$ et enfin à $\mathcal{O}(1)$.

Dans cette dernière section, nous avons présenté des procédures pour simuler $\mathrm{Bin}(n, k/n)$ en utilisant moins de k appels à une procédure générant un sous-ensemble binomial. La meilleure complexité obtenue de $\mathcal{O}(k)$ utilise l'essentiel des techniques présentées dans ce chapitre, et dépend fortement de notre construction combinatoire qui permet de générer le nombre de points fixes d'une permutation de taille n en temps constant (en moyenne), et ce pour tout n.

Chapitre 5

Maintenabilité et simulabilité sans accès à la taille

Sommaire

Ce chapitre explore la maintenance de graphes sans accès à la taille, et en particulier une généralisation du problème soulevé par le chapitre précédent : la simulation en aveugle.

Dans une première partie, la maintenance de distributions de graphes aléatoires est explorée sur certaines familles dans le modèle de décentralisation où on suppose l'accès à une primitive `RandomVertex()`, mais pas à la taille courante du graphe. Cette partie s'ouvre sur la maintenance des graphes k-sortants, liant les résultats des chapitres 3 et 4. Dans un second temps, nous explorons la maintenabilité des graphes d'Erdős–Rényi, avec p fixe, dont l'étude avait été faite au chapitre 1 en supposant la connaissance de n, la taille du graphe. Cette distribution de graphes aléatoires s'avère impossible à maintenir sans avoir accès à la taille (proposition 5.1).

Pour le modèle des graphes k-sortants, *la conversion* des algorithmes de maintenance dans un modèle où la taille n'est pas accessible fait intervenir explicitement le fait qu'une distribution particulière (loi $\mathrm{Bin}(n, k/n)$) doit être simulée en n'ayant

qu'une connaissance parcellaire de ces paramètres. Cette simulation est rendue possible par les différents algorithmes présentés au chapitre 4. Le fait que, d'après la première section, le modèle $G(n,p)$ n'est pas maintenable sans accès à la taille, nous indique que la distribution particulière $\mathrm{Bin}(n,p)$ est non simulable dans ce modèle (autrement, la conversion aurait elle aussi été possible).

La seconde partie de ce chapitre étudie une généralisation de cette notion de simulabilité de lois, que nous appelons problème de la *simulation en aveugle*, qui y est présenté et résolu. Le problème consiste à caractériser les familles de distributions de probabilités sur les entiers $(\rho_n)_{n\in\mathbb{N}}$ pour lesquels il est possible de simuler la distribution ρ_n en ayant uniquement accès à un générateur [1] d'uniformes sur un ensemble de taille n, e.g. `RandomVertex()`. Une caractérisation complète des familles simulables (dont le support peut être infini) est donnée dans cette partie.

La troisième partie met en lumière les liens existants entre la simulabilité de lois comme décrit dans la seconde partie, et la maintenance de distributions de graphes aléatoires étudiée jusqu'à présent. Dans les deux modèles de graphes aléatoires étudiés au début de ce chapitre, la maintenabilité va de pair avec la simulabilité de la distribution des degrés des nœuds, pouvant nous amener à conjecturer que c'est tout le temps le cas. Nous verrons enfin dans cette section que cette conjecture est fausse : il existe des distributions de graphes maintenables possédant une distribution des degrés non simulable.

5.1 Maintenance sans accès à la taille

Nous explorons dans cette section la maintenance de graphes aléatoires dans notre modèle de décentralisation où l'accès à la taille n du graphe courant est impossible. Suivant la distribution considérée une borne inférieure est nécessaire pour que le graphe de départ soit bien défini, par exemple :

- $n \geq 2$ pour les couplages parfaits
- $n \geq k+1$ pour les graphes k-sortants,
- $n \geq |\mu|+1$ pour les graphes μ-sortants,
- $n \geq 1$ pour les autres modèles étudiés ou donnés en exemple, comme les graphes d'Erdős–Rényi ou le multigraphe de paires.

L'unique possibilité pour *améliorer* cette borne inférieure sur n est d'apprendre l'existence de sommets du graphe, notamment à l'aide de la primitive globale `RV`. Nous rappelons que nous supposons que les sommets peuvent être appris de deux manière : soit directement grâce à `RV`, soit en *questionnant* le voisinage d'un sommet dont on connaît déjà l'existence.

La complexité, en nombre de pas de calcul, d'un algorithme de mise à jour est au minimum le nombre de sommets que l'algorithme visite au cours d'une exécution.

1. Comme nous avons vu au chapitre précédent, si seulement la simulabilité est concernée, on peut supposer de manière équivalente que l'on a accès directement à un générateur d'uniformes sur $[\![n]\!]$.

Dans l'objectif de concevoir des algorithmes de maintenance efficaces, cette complexité doit rester la plus faible possible (une complexité sous linéaire est préférable). Cette contrainte nous indique que pour maintenir efficacement des graphes aléatoires de manière décentralisée, peu de connaissances sur la taille du graphe pourront être acquises *généralement*.

Parmi les distributions déjà présentées, les circuits uniformes [2] et les couplages parfaits n'ont nullement besoin de la taille courante pour être maintenus.

5.1.1 Maintenance des graphes k-sortants

La maintenance des graphes k-sortants a été explorée au chapitre 3 en supposant l'accès à la taille du graphe. Le chapitre 4 montre en outre comment simuler efficacement la distribution $\mathrm{Bin}(n, k/n)$ nécessaire à la transposition des algorithmes dans le modèle étudié dans ce chapitre, c'est-à-dire sans connaître la taille du graphe.

Algorithmes de mise à jour naturels

Les algorithmes les plus simples présentés, appelés suppression et insertion naturelle, ne font intervenir la taille du graphe qu'au cours de l'insertion d'un sommet en générant la taille de l'ensemble des prédécesseurs du nouveau sommet comme une variable binomiale. La distribution $\mathrm{Bin}(n, k/n)$ à générer peut être simulée dans notre modèle par l'algorithme 25, Bink-Amélioré, présenté au chapitre précédent, en moins de $5{,}2k - 1$ tirages en moyenne (asymptotiquement, pour n grand). De plus, cet algorithme récupère un sous-ensemble de l'ensemble des sommets de même cardinalité. Donc en $\mathcal{O}(k)$ appels en moyenne, lorsque n est grand, la génération de la variable binomiale et l'appel à la fonction `RandomSubset()` peuvent être effectués.

Cette simulation nous donne les coûts suivants pour la maintenance des graphes k-sortants sans avoir accès à la taille :

- Suppression : coût moyen asymptotique de k ; l'algorithme de suppression Sup-Nat-k-sortant page 75 n'utilise pas en effet la taille du graphe.
- Insertion : coût moyen asymptotique de $\mathcal{O}(k)$; asymptotiquement en moyenne, k premiers appels sont nécessaire pour construire les successeurs du nouveau sommet et $5{,}2k - 1$ appels afin de construire les prédécesseurs en simulant la distribution $\mathrm{Bin}(n, k/n)$.

Autres algorithmes de mise à jour

Tous les autres algorithmes de mise à jour pour les graphes k-sortants uniformes utilisent explicitement la taille du graphe. Pour les algorithmes d'insertion, la distribution $\mathrm{Bin}(n, k/n)$ est simulée à une reprise, à chaque fois pour obtenir la taille du voisinage entrant du nouveau sommet.

2. De manière plus générale, les permutations aussi, en les considérant en tant que graphes orientés avec boucles. Par exemple l'opération insérer après, présentée § 4.3.3, est purement locale.

En dehors de cette génération, les algorithmes de suppression et d'insertion ont tous en commun de restreindre l'utilisation de la taille du graphe à la génération de variable de Bernoulli. Ces variables aléatoires sont toutes de la forme $\mathrm{Ber}(a/(n-b))$ où $n = |V|$ pour les insertions et $n = |V| - 1$ pour les suppressions, $0 \leq a \leq n - b$ et $0 \leq b \leq n - 1$; plus précisément, on retrouve :

- **Sup-Succ-k-sortant** : Bernoulli($|S|/|V'|$) ; S est un ensemble connu de taille au plus $k - 1$. On pourra noter que S ne contient pas u.
- **Sup-Pred-k-sortant** : Bernoulli($|C|/(|V'|-k)$) ; C est un ensemble connu, qui au maximum est de taille $n - 1 - k$. L'ensemble C est de plus disjoint de $N_G^+(u_i)$.
- **Ins-Succ-k-sortant** : Bernoulli($|C|/(|V| - |S|)$) ; C est un ensemble connu de taille au plus k, S est aussi connu et de taille au plus $k - 1$. Par définition C et S sont disjoints.
- **Ins-Pred-k-sortant** :
  - Bernoulli($|Y|/(|V|-i)$) ; Y est un ensemble connu de taille au plus $k-1$ et i prend la valeur $n-1$ au maximum. On notera qu'un ensemble de taille i est connu au moment de tirer la variable, l'ensemble W dans l'algorithme, qui de plus est disjoint de Y.
  - Bernoulli($k/|V|$) ; $N_G^+(X)$ au moment de l'appel est connu.

De manière générale, générer une variable suivant la loi $\mathrm{Ber}(a/(n - b))$ avec $a \leq n - b$ et $n - b \geq 1$ à partir d'un générateur sur un ensemble V de taille n n'est pas difficile. Pour cela, il suffit de générer des uniformes afin d'acquérir un ensemble A de a éléments distincts et un ensemble B de b éléments distincts de telle sorte que $A \cap B = \emptyset$. On exécute ensuite un algorithme par rejet en recommençant lorsque les tirages tombent dans B, et on renvoie 0 si l'élément $x \in V \setminus B$ retenu au final n'est pas dans A et 1 sinon. Une telle simulation nécessitera en moyenne $a + b + 1$ tirages lorsque a et b sont constants et n tend vers l'infini : $a + \mathcal{O}(1/n)$ tirages en moyenne seront utilisés pour construire A, $b + \mathcal{O}(1/n)$ pour construire B et enfin $1 + \mathcal{O}(1/n)$ tirage en moyenne sera utilisé pour obtenir un élément de $V \setminus B$.

Dans notre situation, un tel processus de simulation n'est pas nécessaire. En effet, dans chacun des cas où une loi de Bernoulli doit être générée, un ensemble de taille a est déjà connu, un autre ensemble de taille b est lui aussi connu, et les deux ensembles ont une intersection vide. Donc, un unique appel est asymptotiquement suffisant afin de simuler les générateurs de Bernoulli présents dans les algorithmes étudiés.

Nous arrivons donc à cette conclusion relativement surprenante que tous les algorithmes de mise à jour présentés possèdent le même coût en nombre d'appels à `RV` lorsque l'on ne connaît pas la taille du graphe et devons simuler les fonctions l'utilisant. En effet, pour chaque appel potentiel à `RV` économisé, un tirage de Bernoulli était utilisé.

5.1.2 Non-maintenabilité des graphes d'Erdős–Rényi

Nous avons déjà vu que la distribution des graphes d'Erdős–Rényi peut être maintenue très simplement en utilisant la taille du graphe lors de l'opération d'insertion. Nous prouvons ici que sans cette connaissance, il n'est pas possible de préserver la distribution lors de l'insertion de nouveaux nœuds.

Nous donnons un résultat légèrement plus général. Quelque soit la borne inférieure $a \geq 1$ choisie sur le nombre de sommets présents, la distribution des graphes $G(n,p)$ avec au minimum a sommets est non maintenable.

Proposition 5.1. *Pour p fixe, la distribution $G(n,p) = (\rho_V)_{V \subset \Omega, |V| \geq a}$ n'est pas maintenable sans connaître la taille du graphe dans notre modèle RandomVertex.*

Preuve Supposons qu'il existe un algorithme d'insertion $\mathcal{A}$ qui préserve la distribution $G(n,p)$.

Soit $G \sim G(V,p)$ un graphe de taille au moins a, V son ensemble de sommets et $n = |V|$. Nous allons analyser le comportement de l'algorithme $\mathcal{A}$ sur G.

Dans notre modèle, $\mathcal{A}$ ne connaît ni V ni n. Rappelons que lorsqu'un algorithme apprend l'identité d'un sommet u, il peut parcourir *librement* la composante connexe de u dans le but d'acquérir l'identité d'autres sommets du graphe. Si un sommet v de la composante connexe de u est ultérieurement tiré par l'algorithme, en retour de `RV`, *aucune* nouvelle connaissance du graphe dans son ensemble n'est acquise [3]. Par définition, l'algorithme ne pourra ajouter ou enlever des arêtes au graphe courant qu'*entre* des sommets dont il a appris l'identité.

Comme par hypothèse le nombre de sommets du graphe courant ne peut être inférieur à a, un algorithme de mise à jour peut appeler la primitive `RandomVertex()` jusqu'à acquérir l'identité de a sommets distincts du graphe, tout en étant certain que ce processus termine avec probabilité 1. Lorsque l'algorithme acquiert $\ell \geq a$ sommets en parcourant les composantes connexes des sommets retournés par `RandomVertex()`, il a potentiellement acquis l'identité de tous les sommets du graphe.

Soit A un sous-ensemble de V de taille a. Pour tout graphe $g \in \mathcal{G}_A$, il doit exister un nombre $k(g)$ tel que $\mathcal{A}$ s'arrête avec probabilité au moins $1/2$ après avoir tiré $k(g)$ sommets de g, quelque soit la séquence des tirages. Si un tel nombre n'existe pas pour un certain graphe g de $\mathcal{G}_A$, alors l'algorithme ne préserve pas la distribution $G(n,p)$ car sa probabilité d'arrêt sur g n'est pas 1. On note $k = \max\{k(g) \mid g \in \mathcal{G}_A\}$. Nous avons donc qu'après avoir tiré k sommets, tous appartenant à un ensemble A de taille a, l'algorithme $\mathcal{A}$ s'arrête avec probabilité au moins $1/2$.

Donc la probabilité que $\mathcal{A}$ s'arrête en ayant pris connaissance d'au maximum a sommets est au moins $(a/n)^k/2$.

3. Une information *probabiliste* est néanmoins acquise : l'évènement « tomber sur v » a probabilité $1/n$ de se produire.

Remarquons à présent que la probabilité que l'ensemble fixé A soit *déconnecté* du reste du graphe G, *i.e.* il n'existe pas d'arêtes entre A et $V \setminus A$ dans G, est $(1-p)^{a(n-a)}$.

Soit $u \in \Omega \setminus V$, un nouveau sommet à insérer dans G, et $G' = \mathcal{A}(G,u)$ le graphe aléatoire obtenu après insertion de u.

D'après l'argumentation ci-dessus, nous pouvons minorer la probabilité q que l'ensemble $A+u$ soit déconnecté dans G',

$$q \geq (1-p)^{a(n-a)} \frac{1}{2} \left(\frac{a}{n}\right)^k$$

car par définition, $\mathcal{A}$ ne peut ajouter d'arêtes incidentes à des sommets dont il n'a pas acquis l'identité.

Enfin, dans un graphe aléatoire distribué selon la distribution $G(V+u,p)$, l'ensemble $A+u$ est isolé du reste du graphe avec probabilité $(1-p)^{(a+1)(n-a)}$.

Nous concluons en remarquant que, pour n suffisamment grand, $q > (1-p)^{(a+1)(n-a)}$ donc $G' \not\sim G(V+u,p)$ pour V de taille suffisamment grande. L'algorithme $\mathcal{A}$ n'est pas un algorithme d'insertion correct pour la distribution $G(n,p)$, et donc les graphes aléatoires $G(n,p)$ ne sont pas maintenables.

□

5.2 Simulation en aveugle

La question essentielle posée dans cette section est celle du pouvoir de simulation donné aux algorithmes en utilisant la primitive `RV`.

Nous allons caractériser précisément les familles de distributions $(\rho_n)_{n\in\mathbb{N}}$ qui peuvent être simulées dans notre modèle. Ici, simuler la famille $(\rho_n)_{n\in\mathbb{N}}$ consistera à générer une variable aléatoire selon ρ_n lorsque l'ensemble sous-jacent de `RV` est de taille n.

5.2.1 Modèles

Une famille de distributions de probabilité sur l'ensemble des entiers naturels sera dénotée par $\rho = (\rho_n)_{n\geq a}$, où a est supposé être un entier supérieur à 1 que l'on ne spécifiera pas généralement. Dans la suite, lorsque cela ne porte pas à confusion, nous parlerons uniquement de *famille* à la place de famille de distributions de probabilité.

Lorsque $\rho = (\rho_n)_{n\geq a}$ désigne une telle famille, $\mathrm{Supp}(\rho)$ désigne l'union des supports des distributions de ρ, *i.e.* $\mathrm{Supp}(\rho) = \bigcup_{n\geq a} \mathrm{Supp}(\rho_n)$.

Dans la suite, nous tâcherons de conserver les notations suivantes afin de faciliter la lecture : $\rho = (\rho_n)_{n\geq a}$ est la distribution que l'on souhaite simuler, V dénote l'ensemble sous-jacent des tirages aléatoires, $n \geq a$ est la taille de cet ensemble, k est une des valeurs de $\mathrm{Supp}(\rho)$, et $m \in [\![a, n]\!]$ est une borne inférieure sur n acquise par l'algorithme (généralement le nombre d'éléments de V distincts vu jusqu'à présent).

Toutes les familles de distributions $\rho = (\rho_n)_{n\in\mathbb{N}}$ que nous manipulerons ici sont supposées *simulables* dans un modèle conventionnel de simulation, en ayant la connaissance de n. Plus précisément, nous supposons qu'il existe une machine de Turing probabiliste, prenant en entrée n, et dont la valeur de retour est distribuée selon ρ_n. Cette définition est équivalente à dire que la fonction $\rho : \mathbb{N} \times \mathbb{N} \to [0,1]$, telle que $\rho(n,k) = \rho_n(k)$, est une fonction calculable sur les réels (cf. [56]).

Cette garantie nous indique qu'il est possible de donner des approximations des valeurs $\rho_n(k)$ à précision arbitraire. Il est donc aussi possible de simuler des lois ne faisant intervenir que des opérations arithmétiques de base (sommes, soustractions, division, multiplication), car nous pouvons approximer à précision arbitraire les probabilités intervenant dans la loi (à partir des approximations des différents $\rho_n(k)$ qui sont impliqués).

Nous ne rappellerons pas ces hypothèses par la suite.

Notation Nous introduisons la notation suivante $\eta(\rho,k)$ pour désigner la première valeur de n où k apparaît dans le support de ρ_n, pour une certaine famille de lois $\rho = (\rho_n)_{n\geq a}$, *i.e.* $\eta(\rho,k) = \min\{n \geq a \mid k \in \mathrm{Supp}(\rho_n)\}$.

Ci-après le théorème principal de cette section.

Théorème 5.2. *Une famille $\rho = (\rho_n)_{n\geq a}$ de distributions de probabilité est simulable, si et seulement si, il existe une fonction calculable $f : Supp(\rho) \to \mathbb{N}$ telle que pour tout $k \in Supp(\rho)$, et pour tout $n \geq \max\{2,\eta(\rho,k)\}$, $\rho_n(k) \geq 1/n^{f(k)}$.*

Nous commençons par définir formellement le modèle d'aléa dans lequel la simulation est effectuée, puis nous fournissons une preuve que la condition donnée est nécessaire pour toute famille ρ (théorème 5.6). Le fait qu'elle soit aussi suffisante est donné en deux temps, d'abord exclusivement pour les familles à supports finis (théorème 5.11), puis enfin pour les familles à supports infinis (théorème 5.12).

Modèle de tirages

Nous formalisons dans ce paragraphe le problème de simulation de loi à partir de tirages d'uniformes sur un ensemble V. Dans ce premier modèle, nous ferons l'hypothèse que les algorithmes n'ont pas accès à d'autres sources d'aléa que la source de tirages sur V.

Nous supposons l'existence d'un ensemble sous-jacent V de taille n, et l'existence d'une fonction `draw()` permettant de tirer aléatoirement et uniformément des éléments de V. Les appels successifs à la fonction renvoient des résultats indépendants. Cette fonction est le pendant de notre primitive `RV` dans ce modèle où n'apparaissent pas de graphes aléatoires.

Comme les algorithmes de simulation n'ont pas accès à V, le tout premier appel à `draw()` ne fournit (quasiment) aucune information. Au second appel, il existe seulement deux possibilités de *séquences* en prenant le point de vue de l'algorithme :

soit le tirage renvoie le même élément que le premier (ce qui arrive avec probabilité $1/n$), soit un nouvel élément est obtenu (avec probabilité $1 - 1/n$). Le troisième tirage produit potentiellement 4 séquences différentes : 111, 112, 121, 122, 123, en identifiant chaque élément par l'ordre de leur apparition. Après quatre tirages, il existe 15 telles séquences. Notons que toutes ces séquences potentielles de longueur ℓ ne sont réalisables que dans le cas où n est au moins égal à ℓ.

Nous avons donc qu'après m appels à la fonction `draw()`, la totalité de l'information accumulée par l'algorithme est codée dans la séquence de tirages, *i.e.* une séquence de m entiers non nuls où le i-ème élément est au maximum un de plus que le maximum des $i-1$ premiers. De plus, quelque soit m, la séquence ne pourra contenir plus de $n = |V|$ éléments distincts. Nous appellerons une telle séquence s une *séquence de tirages* sur n, et nous noterons $|s|$ sa longueur (ou taille) et $||s||$ le nombre d'éléments différents la formant, *i.e.* le maximum de la séquence par définition.

Définition 5.1. Une séquence de tirages s sur n est une séquence d'entiers non nuls de longueur $|s|$ telle que $s_1 = 1$, pour tout $1 \leq i \leq |s|$, $s_i \leq \max\{s_j | j \in [\![i-1]\!]\} + 1$ et $||s|| = \max\{s_i | i \in [\![|s|]\!]\} \leq n$.

Remarque Pour les longueurs m au plus égales à n, le nombre de séquences de tirages sur n de longueur m est compté par les nombres de Bell ou nombres exponentiels ([87], Oeis/A000110), qui correspondent, entre autres, au nombre de façons de partitionner un ensemble de m éléments étiquetés.

Il existe une bijection simple pour s'en rendre compte : dans une séquence s sur n avec $m = |s|$, s_i indique le numéro de l'ensemble de l'élément $i \in [\![m]\!]$ dans le partitionnement. Cette correspondance nous indique que les séquences de tirages sont en bijection avec les partitions en au plus n parts. Ces partitions sont comptées par une somme partielle de nombres de Stirling de seconde espèce $\sum_{k=1}^{n} \left\{ {m \atop k} \right\}$, voir Oeis/A102661.

Soit $\mathcal{S}^n = \bigcup_{m \geq 1} \mathcal{S}^n_m$ l'ensemble de toutes les séquences de tirages sur n, où $\mathcal{S}^n_m$ est l'ensemble de telles séquences de longueur m. On remarque $\mathcal{S}^n \subset \mathcal{S}^{n+1}$.

Nous notons $\mathcal{S} = \bigcup_{n \geq 1} \mathcal{S}^n$ l'ensemble de toutes les séquences de tirages potentielles.

Comme les algorithmes n'ont pas connaissance de l'ensemble V, les décisions pouvant être prises ne pourront dépendre que de la séquence de tirages obtenue via les tirages sur V. De plus, comme nous faisons l'hypothèse que les algorithmes de simulation sont *déterministes* (en dehors des appels à `draw()`), les seules possibilités après la *lecture* d'une certaine séquence de tirages sont soit de s'arrêter et retourner une valeur, soit de continuer le processus de tirages. Nous pouvons donc associer à chaque algorithme $\mathcal{A}$ une fonction calculable $f_{\mathcal{A}} : \mathcal{S} \to R \cup \{\perp\}$ où R est l'ensemble des valeurs de retour pour $\mathcal{A}$. Cette fonction décrit partiellement le comportement de $\mathcal{A}$, en désignant les séquences de tirages s obtenues à partir d'appels à `draw()` où $\mathcal{A}$ s'arrête :

— $f_{\mathcal{A}}(s) = k \in R$ si et seulement si $\mathcal{A}$ retourne k après avoir tiré la séquence s ;
— $f_{\mathcal{A}}(s) = \perp$ sinon, *i.e.* si et seulement si $\mathcal{A}$ ne peut pas atteindre la séquence s (il s'est arrêté sur un préfixe de s) ou $\mathcal{A}$ peut atteindre s mais continue le processus de tirages.

Comme un algorithme ne s'arrête qu'une seule fois, une conséquence directe est que pour chaque séquence de tirages $s \in \mathcal{S}$, si $f_{\mathcal{A}}(s) \neq \perp$, alors pour tout préfixe propre s' de s, $f_{\mathcal{A}}(s') = \perp$ et pour toute séquence de tirages $s'' \in \mathcal{S}$ dont s est un préfixe propre, $f_{\mathcal{A}}(s'') = \perp$.

À chaque séquence de tirages s sur n, nous pouvons associer une probabilité $\mathbb{P}_n(s)$ d'obtenir la séquence à partir d'appels à la primitive `draw()` lorsque les tirages sont uniformes et indépendants sur l'ensemble V.

Lemme 5.3. *La probabilité d'obtenir la séquence de tirages s est*

$$\mathbb{P}_n(s) = \frac{(n)_{||s||}}{n^{|s|}}.$$

Preuve Après avoir tiré i éléments différents, un nouvel élément est tiré avec probabilité $(n-i)/n$ et chacun des éléments déjà tirés a probabilité $1/n$ d'être tiré de nouveau.

Comme par définition les tirages sont indépendants entre eux, la probabilité d'obtenir une certaine séquence s est

$$\frac{n}{n} \cdot \frac{n-1}{n} \cdots \frac{n-||s||+1}{n} \frac{1}{n^{|s|-||s||}} = \frac{(n)_{||s||}}{n^{|s|}}.$$

□

Définition 5.2. La probabilité $\mathbb{P}_n(\mathcal{A},k)$, sous l'hypothèse que l'ensemble sous-jacent V est de taille n, que l'algorithme $\mathcal{A}$ retourne la valeur k, est

$$\sum_{s \in \mathcal{S}^n | f_{\mathcal{A}}(s)=k} \mathbb{P}_n(s).$$

Nous sommes à présent prêts à définir le comportement d'un algorithme de simulation correct pour une certaine famille de distributions de probabilité ρ.

Définition 5.3. Un algorithme $\mathcal{A}$ simule la famille $\rho = (\rho_n)_{n \geq a}$, si pour tout n supérieur à a, pour tout k appartenant à $\mathrm{Supp}(\rho_n)$, nous avons $\mathbb{P}_n(\mathcal{A},k) = \rho_n(k)$.

Remarque Si $\mathcal{A}$ simule la famille ρ, alors du fait que ρ_n soit une distribution de probabilité, nous avons $\sum_{k \in \mathrm{Supp}(\rho_n)} \rho_n(k) = 1$ et donc tout algorithme de simulation termine avec probabilité 1 pour tout n.

Une famille $\rho = (\rho_n)_{n \geq a}$ est dite *simulable* s'il existe un algorithme $\mathcal{A}$ qui simule ρ. Une condition évidement nécessaire afin que $\mathcal{A}$ puisse simuler la famille ρ est, qu'après avoir lu la séquence s, les éléments retournés par $\mathcal{A}$ soient contenus dans $\mathrm{Supp}(\rho_{||s||})$. Il est facile de s'en convaincre : si $\mathcal{A}$ retourne un élément x en dehors de $\mathrm{Supp}(\rho_{||s||})$ après avoir tiré la séquence s, alors $\mathbb{P}_{||s||}(\mathcal{A}, x) \geq \mathbb{P}_{||s||}(s) > 0$, ce qui ne correspond point à $\rho_{||s||}(x) = 0$.

Nous terminons cette section par un lemme simple, mais contraignant fortement les familles pouvant être simulées. Seules sont simulables, celles dont les supports sont inclus les uns dans les autres.

Lemme 5.4. *Si $(\rho_n)_{n \geq a}$ est une famille simulable, alors pour tout $n \geq a$, $Supp(\rho_n) \subset Supp(\rho_{n+1})$.*

Preuve Soit $\mathcal{A}$ un algorithme simulant $\rho = (\rho_n)_{n \geq a}$, une famille de distributions de probabilité donnée. Supposons que la proposition ne soit pas vérifiée, *i.e.* il existe $n \geq a$ et un élément k appartenant à $\mathrm{Supp}(\rho_n)$ mais pas à $\mathrm{Supp}(\rho_{n+1})$.

Comme $k \in \mathrm{Supp}(\rho_n)$ et $\mathcal{A}$ simule ρ, $\mathbb{P}_n(\mathcal{A}, k) = \rho_n(k) > 0$. Donc, par définition de $\mathbb{P}_n(\mathcal{A}, k)$, il existe $s \in \mathcal{S}^n$ tel que $f_{\mathcal{A}}(s) = k$. Comme $\mathcal{S}^n \subset \mathcal{S}^{n+1}$, $\mathbb{P}_{n+1}(\mathcal{A}, k) \geq \mathbb{P}_{n+1}(s) > 0$.

Par conséquence $\mathbb{P}_{n+1}(\mathcal{A}, k) \neq \rho_{n+1}(k)$ et donc $\mathcal{A}$ ne simule pas ρ.

□

Remarque Si $\rho = (\rho_n)_{n \geq 1}$ est simulable, alors $\rho_1 = \delta_x$ est une Dirac en x, *i.e.* $|\mathrm{Supp}(\rho_1)| = 1$, car il ne peut exister qu'une seule séquence ne contenant que des 1 sur laquelle un algorithme donné s'arrête.

Par la suite, nous ne considérerons que des familles valides, au sens du lemme 5.4.

Modèle de tirages avec aléa local

Dans ce qui précède, nous nous sommes intéressés à des algorithmes qui ne possédaient pas de source d'aléa (interne), exception faite du processus de tirages. Dans l'objectif de concevoir des algorithmes de simulation, et spécialement si l'efficacité est visée, cette hypothèse apparaît trop restrictive. Pour cette raison, lors de la description des algorithmes de simulation, nous ferons l'hypothèse qu'ils ont accès à une *source d'aléa local* indépendante des tirages sur V. Cette source est matérialisée sous la forme d'une primitive `flip()` qui renvoie des bits uniformes et indépendants, et les appels à la fonction sont indépendants des résultats du processus de tirage.

Nous définissons une séquence *draw-flip* s comme une séquence de tirages entrelacée de bits marqués (pour les différencier du chiffre 1), *i.e.* $s \in \mathbb{N} \cup \{\underline{0}, \underline{1}\}$. De telles séquences représentent les résultats alternants d'appels aux primitives `draw()` et `flip()`. Ainsi, le sous-mot constitué des tirages sur V doit suivre les contraintes imposées à une séquence de tirages comme définie précédemment, et il n'y a pas de contraintes en ce qui concerne la sous-séquence des bits. Une telle séquence s a

pour longueur (totale) $|s|$, contient $|s|_f$ bits, et $|s|_d = |s| - |s|_f$ entiers dont $||s||$ sont distincts. L'ensemble de toutes ces séquences contenant au plus n entiers différents, est noté $\mathcal{F}^n$ et $\mathcal{F} = \bigcup_{n\geq 0} \mathcal{F}^n$ correspond à l'ensemble de telles séquences.

Comme dans le modèle précédent, nous associons à chaque algorithme de simulation $\mathcal{A}$ une fonction calculable $f_{\mathcal{A}} : \mathcal{F} \to R \cup \{\perp\}$ décrivant partiellement le comportement de l'algorithme, où R correspond à l'ensemble des valeurs de retours de $\mathcal{A}$. Pour $s \in \mathcal{F}$, nous avons :

— $f_{\mathcal{A}}(s) = k \in R$ si et seulement si $\mathcal{A}$ retourne k en ayant obtenu la séquence s à partir des primitives ;

— $f_{\mathcal{A}}(s) = \perp$ sinon, *i.e.* si et seulement si s est impossible à atteindre ou si $\mathcal{A}$ continue à appeler une des primitives après avoir obtenu la séquence s.

Une séquence s est inatteignable, si s possède un préfixe propre s' tel que $f_{\mathcal{A}}(s') \in R$, ou si l'entrelacement de s ne correspond pas au comportement de $\mathcal{A}$, par exemple $s = 1$ alors que $\mathcal{A}$ commence par appeler la fonction `flip()`.

La probabilité $\mathbb{P}_n(s)$ d'obtenir une séquence particulière s, en supposant que l'ensemble sous-jacent est de taille n, se calcule aisément

$$\mathbb{P}_n(s) = \frac{1}{2^{|s|_f}} \cdot \frac{(n)_{||s||}}{n^{|s|_d}}.$$

Nous obtenons dans ce modèle une définition équivalente pour la probabilité de retourner une certaine valeur k, $\mathbb{P}_n(\mathcal{A}, k) = \sum_{s\in\mathcal{F}^n \mid f_{\mathcal{A}}(s)=k} \mathbb{P}_n(s)$; les même notions de simulabilité sont posées dans ce modèle en se référant à la définition ci-avant de $\mathbb{P}_n(\mathcal{A}, k)$.

Les deux modèles présentés, bien qu'ayant des pouvoirs de simulation équivalents la plupart du temps, ne sont pas interchangeables dans le cas général. Par exemple, si n est autorisé à valoir 1, n'importe quel algorithme dans le modèle précédent est purement déterministe lorsque n vaut 1 et donc seulement des lois de Dirac peuvent être simulées ; au contraire, les algorithmes ayant accès à des bits uniformes peuvent simuler d'autres distributions lorsque $n = 1$.

Néanmoins, dès que $n \geq 2$, les appels à la primitive `flip()` peuvent être simulés uniquement à partir d'appels à la fonction `draw()`, et ce sans avoir accès à V. Afin de simuler des bits i.i.d uniformes nous pouvons utiliser une méthode bien connue, parfois appelée *extracteur de Von Neumann*, et qui apparaît déjà en 1951 [90]. La méthode consiste à générer des $\mathrm{Ber}(1/2)$ indépendantes à partir de $\mathrm{Ber}(p)$ indépendantes, où $p \in]0, 1[$. Pour cela, deux $\mathrm{Ber}(p)$ sont générées jusqu'à qu'elle soient différentes, puis le résultat de la première est renvoyé. Dans notre cas, un événement de probabilité $1/n$ peut facilement être généré en tirant deux éléments de V, et en renvoyant 1 s'ils sont identiques. Notons qu'il existe des méthodes plus efficaces, et une simple itération de l'extracteur de Von Neumann [78] atteint la borne d'entropie optimale pour la source donnée .

La proposition suivante explicite l'équivalence entre les modèles présentés.

Proposition 5.5. *Une famille $\rho = (\rho_n)_{n\geq 1}$ est simulable dans notre premier modèle, i.e. uniquement à partir d'appels à* `draw()`*, si et seulement si ρ est simulable dans notre second modèle (avec aléa local) et $\rho_1 = \delta_x$ est une Dirac en x.*

Preuve Remarquons tout d'abord, que n'importe quelle famille simulable en utilisant uniquement la primitive `draw()`, l'est aussi en ajoutant la primitive `flip()`, et la distribution simulée est bien une Dirac lorsque $n = 1$.

Supposons à présent $\rho_1 = \delta_x$ et $\rho = (\rho_n)_{n\geq 1}$ est simulable dans notre second modèle ; soit $\mathcal{A}$ un algorithme la simulant. Comme $\mathcal{A}$ simule ρ, $\mathbb{P}_1(\mathcal{A}, x) = \rho_1(x) = 1$, et il doit exister des séquences *draw-flip* s appartenant à $\mathcal{F}^1$ telles que $f_{\mathcal{A}}(s) = x$. Soit s_0 l'une d'entre elle, par exemple la plus petite dans l'ordre lexicographique. On note w la sous-séquence constituée des bits de s_0.

Nous proposons tout d'abord de remplacer $\mathcal{A}$ par un autre algorithme de simulation $\mathcal{A}'$, qui simule aussi ρ dans notre modèle *draw-flip*. L'algorithme $\mathcal{A}'$ est identique à $\mathcal{A}$, mis à part qu'on applique un ou-exclusif avec w sur les $|w|$ premiers bits renvoyés par `flip()`. Quelque soit w, les bits pris en compte par $\mathcal{A}'$ sont uniformes et indépendants dès lors que les résultats des `flip()` sont eux même uniformes et indépendants. Donc la distribution des valeurs de retour de $\mathcal{A}'$ est identique à celle de $\mathcal{A}$, et donc $\mathcal{A}'$ simule bien ρ.

Nous remarquons que $\mathcal{A}'$ s'arrête et retourne x sur une séquence s où tous les flips ont renvoyé 0, et dont tous les éléments tirés sur V sont identiques. Partant de cette constatation, nous allons simuler $\mathcal{A}'$ dans le modèle *draw* en simulant les bits uniformes via des tirages sur V, avec $n = |V|$, de telle sorte que la simulation ne produise que des 0 lorsque V est de taille 1 (au contraire de l'extracteur de Von Neumann qui boucle si $p = 1$).

Nous proposons l'extracteur de bits suivant : un élément b est d'abord identifié par un premier appel à `draw()`, puis,

1. l'extracteur tire un élément sur V, et si c'est b, il retourne 0 ; sinon on a identifié un second élément $b' \neq b$;
2. l'extracteur tire des éléments sur V tant qu'il sont égaux à b ; si le dernier élément tiré est b', il retourne 1, sinon on recommence à l'étape 1.

Nous avons lorsque $n \geq 2$, et pour chaque tentative d'extraction,
— $\mathbb{P}_n(\text{renvoyer } 0) = 1/n$
— $\mathbb{P}_n(\text{renvoyer } 1) = (1 - 1/n) \cdot 1/(n-1) = 1/n$
— $\mathbb{P}_n(\text{recommencer}) = 1 - 2/n$

Donc l'extracteur termine avec probabilité 1, et fonctionne comme un algorithme par rejet, d'où $\mathbb{P}_n(\text{renvoyer } 0) = \mathbb{P}_n(\text{renvoyer } 1) = 1/2$.

Notons, lorsque V est de taille 1, $\mathbb{P}(\text{renvoyer } 0) = 1$.

La simulation de $\mathcal{A}'$ en remplaçant les appels à `flip()` par l'extracteur décrit fournit un algorithme $\mathcal{A}''$ qui n'utilise que des tirages sur V. L'algorithme retourne toujours x lorsque $n = 1$ car $\mathbb{P}_1(\mathcal{A}'', x) = \mathbb{P}_1(s_0) = 1$ et a le même comportement probabiliste que $\mathcal{A}'$ lorsque $n \geq 2$.

□

Remarque Dans la précédente preuve, nous n'avons pas cherché à améliorer l'efficacité de l'extracteur car seule la simulabilité était recherchée.

Exemple 5.1. Nous avons déjà rencontré plusieurs familles de distributions simulables en aveugle :

- Le chapitre précédent présente plusieurs algorithmes afin de simuler en aveugle les familles de distributions $(\mathrm{Bin}(n, 1/n))_{n\geq 1}$ et $(\mathrm{Bin}(n, k/n))_{n\geq k}$.
- Au cours du chapitre précédent, page 114, une procédure permettant de générer une variable suivant la distribution $\mathrm{Unif}(\llbracket |V| \rrbracket)$ est donnée, permettant ainsi de simuler la famille correspondante $(\mathrm{Unif}(\llbracket n \rrbracket))_{n\geq 1}$.
- Au début de ce chapitre au § 5.1.1, une méthode est fournie afin de simuler la famille de distributions $(\mathrm{Ber}(a/(n-b)))_{n\geq a+b}$ avec $a \geq 1$ et $b \geq 0$.

■

Dans ce qui suit, nous considérerons que les algorithmes sont basés sur notre second modèle de tirages, avec aléa local, et que les distributions de probabilités usuelles y sont implémentées via la primitive `flip()`. En se basant sur la proposition 5.5, il est dorénavant facile de déterminer si une distribution particulière peut être simulée sans utiliser d'aléa local. Il faut que ρ_1 soit une Dirac ou que la cardinalité de l'ensemble sous-jacent V ne soit pas autorisée à être égale à 1.

5.2.2 Résultat de non simulabilité

Nous savons déjà que certaines familles, appelées *invalides*, ne sont pas simulables car elles ne possèdent pas de supports inclus les uns dans les autres. Nous pouvons de plus caractériser, parmi les familles valides, une classe de familles de distributions non simulables. Ces familles $(\rho_n)_{n\geq a}$ possèdent des probabilités $\rho_n(k)$ qui tendent trop rapidement (*i.e.* superpolynomialement) vers 0 lorsque n tend vers l'infini, et ce pour au moins un élément k faisant partie du support de ρ.

Comme exemple typique de cette situation, montrons pourquoi la famille de lois $\mathrm{Bin}(n, 1/2) = (b_n)_{n\geq 2}$ n'est pas simulable ; on donnera un énoncé plus général plus tard. Supposons qu'il existe un certain algorithme $\mathcal{A}$ simulant $(b_n)_{n\geq 2}$. Comme n peut éventuellement être égal à 2, $\mathcal{A}$ doit s'arrêter et retourner 0 sur des séquences *draw-flip* ne contenant que deux éléments distincts de V. Soit s une telle séquence, α sa longueur, et $\mathbb{P}_n(s) \geq 1/n^\alpha$ la probabilité d'obtenir s. Par conséquent, $\mathbb{P}_n(\mathcal{A}, 0) \geq \mathbb{P}_n(s) \geq 1/n^\alpha > b_n(0) = 1/2^n$ pour n suffisamment grand. Le même type de raisonnement s'étend facilement à $\mathrm{Bin}(n, p)$ pour $n \geq a$, il suffit alors de considérer des séquences contenant au moins a éléments distincts sur lesquelles $\mathcal{A}$ s'arrête.

Nous généralisons l'intuition précédente à n'importe quelle famille de lois $(\rho_n)_{n\geq a}$ dont certaines valeurs du support sont de moins en moins probables, et ce de manière trop rapide lorsque n augmente. Ici, « trop rapide » signifie plus que polynomialement rapidement.

Théorème 5.6. *Si une famille $\rho = (\rho_n)_{n\geq a}$ est simulable, alors il existe une fonction calculable f : Supp$(\rho) \to \mathbb{N}$ telle que pour tout $k \in$ Supp(ρ), et pour tout $n \geq \max\{2,\eta(\rho,k)\}$, $\rho_n(k) \geq 1/n^{f(k)}$.*

Preuve Soit $k \in \mathrm{Supp}(\rho)$ et $m = \eta(\rho,k)$, donc $k \in \mathrm{Supp}(\rho_m)$, et pour tout $n \geq m$ nous avons $\rho_n(k) > 0$ par le principe d'inclusion des supports.

Soit $\mathcal{A}$ un algorithme simulant ρ. Comme $\rho_m(k) > 0$, il existe une séquence s, avec $||s|| \leq m$, telle que $f_{\mathcal{A}}(s) = k$. De plus, nous avons pour tout $n \geq m$,

$$\mathbb{P}_n(\mathcal{A},k) \geq \mathbb{P}_n(s) \geq \frac{1}{2^{|s|_f}} \frac{1}{n^{|s|_d}}$$

Nous avons donc, pour tout $n \geq 2$, $\rho_n(k) = \mathbb{P}_n(\mathcal{A},k) \geq 1/n^{|s|}$.

Enfin, remarquons que pour trouver une telle séquence s, on peut facilement générer toutes les séquences *draw-flip* possibles dans l'ordre lexicographique, simuler $\mathcal{A}$ dessus, jusqu'à ce que $\mathcal{A}$ s'arrête en retournant k ; la fonction f est donc bien calculable.

□

Corollaire 5.7. *Si une famille $\rho = (\rho_n)_{n\geq a}$ est simulable, alors pour tout $k \in$ Supp(ρ), il existe une constante $\beta = \beta(k)$, telle que $\rho_n(k) = \Omega(1/n^\beta)$.*

Remarque Le principe d'inclusion des supports est directement déduit du théorème 5.6.

5.2.3 Critère de simulabilité

Schéma d'algorithmes en deux étapes

Pour $a \geq 1$, dans le but de simuler une famille de lois $\rho = (\rho_n)_{n\geq a}$ valide à supports finis[4], nous proposons un schéma d'algorithmes fonctionnant en deux étapes.

Le schéma est le suivant :

1. dans une première étape, l'algorithme « devine » une valeur m pour n, la taille de V, et cette valeur est *réaliste* dans le sens où $m \in [\![a, n]\!]$;
2. dans une seconde étape, l'algorithme retourne $k \in \mathrm{Supp}(\rho_m)$ avec probabilité p_k^m, ne dépendant que de k et m.

Pour faciliter les notations, on fixe $p_k^m = 0$ pour $k \notin \mathrm{Supp}(\rho_m)$.

La première étape d'un algorithme suivant le schéma décrit se réalise en tirant des éléments uniformes sur V jusqu'à prendre une décision pour m, alors que la seconde étape n'utilise que des tirages de bits uniformes. Nous ne contraignons pas comment la première étape se résout, mais imposons juste que la seconde dépende uniquement de m et non de toute la séquence s tirée à la première étape.

4. Une famille $\rho = (\rho_n)_{n\geq a}$ est à supports finis si pour tout $n \geq a$, $|\mathrm{Supp}(\rho_n)| < \infty$. Toutefois Supp$(\rho)$, l'union des supports, peut être infini.

En particulier, les algorithmes présentés par la suite poseront toujours $m = ||s||$ et ne retourneront que des valeurs de $\mathrm{Supp}(\rho_{||s||})$.

L'idée derrière ce schéma d'algorithmes est que si on *ajuste* convenablement la distribution p_k^m de l'entier retourné après l'estimation "$n = m$", en sommant sur les valeurs possibles de m, nous pourrons obtenir la distribution cible. Lorsque l'estimation est a, l'algorithme n'a pas d'autres choix que de retourner k avec probabilité ρ_a.

Pour un algorithme en 2 étapes, on désigne par *probabilité de première étape* ℓ_m^n, la probabilité d'obtenir m à la fin de la première étape, en supposant $|V| = n$, et par *probabilité de seconde étape* p_k^m, la probabilité de retourner k en ayant choisi m à la première étape.

Il n'est pas difficile de se rendre compte que pour tous les algorithmes suivant ce schéma et simulant une certaine famille ρ, les probabilités p_k^m suivent une récurrence précise dépendant à la fois de ρ et des probabilités ℓ_m^n.

Proposition 5.8. *Soit $\mathcal{A}$ un algorithme en deux étapes simulant la famille $\rho = (\rho_n)_{n \geq a}$, avec probabilité de première étape ℓ_m^n, et probabilité de seconde étape p_k^m. Alors pour tout $m \geq a$, pour tout $k \in Supp(\rho_m)$, p_k^m suit le schéma de récurrence suivant :*

$$p_k^m = \frac{\rho_m(k) - \sum_{j=a}^{m-1} \ell_j^m p_k^j}{\ell_m^m}.$$

Preuve La récurrence découle directement de la formule de probabilité totale pour la variable aléatoire m :

$$\mathbb{P}_n(\mathcal{A},k) = \sum_{m=a}^{n} \ell_m^n p_k^m = \sum_{m=a}^{n-1} \ell_m^n p_k^m + \ell_n^n p_k^n,$$

où $\mathbb{P}_n(\mathcal{A},k) = \rho_n(k)$.

□

Remarque La récurrence fournie est bien initialisée car $\ell_a^a = 1$ (par définition $a \leq m \leq n$) et donc $p_k^a = \rho_a(k)$ pour $k \in \mathrm{Supp}(\rho_a)$. De plus, le schéma de récurrence nous indique aussi que pour tout $m \geq a$, $\sum_{k \in \mathrm{Supp}(\rho_m)} p_k^m = 1$ (par induction sur m).

Notons qu'il n'est pas toujours possible de *compléter* un algorithme produisant une estimation de n par un algorithme en deux étapes, et ce même en connaissant ℓ_m^n. Effectivement, le schéma de récurrence que devrait satisfaire le coefficient p_k^m peut produire des valeurs négatives (et d'autres plus grandes que 1) pour certaines

valeurs de ℓ_m^n. Nous donnons ci-après une condition simple à satisfaire pour que l'algorithme [5] soit **bien défini** suivant la définition suivante.

Définition 5.4. Un algorithme $\mathcal{A}$ en deux étapes, ayant une probabilité de seconde étape p_k^m, est appelé *bien défini* pour $\rho = (\rho_n)_{n\geq a}$ s'il respecte le schéma de récurrence pour ρ et si pour tout $m \geq a$, pour tout $k \in \mathrm{Supp}(\rho_m)$, $\sum_{j=a}^{m-1} \ell_j^m p_k^j \leq \rho_m(k)$.

Remarque Pour que $\mathcal{A}$ corresponde effectivement à un algorithme, il faut certainement $p_k^m \geq 0$ pour $m \geq a$ et $k \in \mathrm{Supp}(\rho_m)$. De surcroît, puisque les p_k^m se somment à 1, $\mathcal{A}$ définit bien un algorithme car p_k^m est effectivement une distribution de probabilité sur $\mathrm{Supp}(\rho_m)$. La simulabilité de la distribution p_k^m se déduit de nos hypothèses sur ρ : les probabilités individuelles p_k^m peuvent être approchées.

De la définition précédente et la preuve de la proposition 5.8, nous déduisons directement la proposition suivante :

Proposition 5.9. *Si un algorithme $\mathcal{A}$ en deux étapes pour ρ est bien défini, alors $\mathcal{A}$ simule ρ.*

Nous terminons cette section par un lemme simple mais qui réduit grandement la difficulté de prouver la positivité des p_k^m (la difficulté essentielle dans de ce type d'algorithme) en proposant une condition plus lâche.

Lemme 5.10. *Soit $\mathcal{A}$ un algorithme en deux étapes pour $\rho = (\rho_n)_{n\geq a}$ avec probabilité de première étape ℓ_m^n, et qui respecte le schéma de récurrence. Si pour tout $n \geq a$, pour tout $k \in Supp(\rho_n)$, $\sum_{m=\eta(\rho,k)}^{n-1} \ell_m^n \leq \rho_n(k)$ alors $\mathcal{A}$ simule ρ.*

Preuve La preuve est une simple induction sur n pour chacune des propositions $\sum_{m=a}^{n-1} \ell_m^n p_k^m \leq \rho_n(k)$ et $0 \leq p_k^n \leq 1$.

Le cas de base $n = a$ correspond à une somme vide et donc l'inégalité sur $\rho_n(k)$ est triviale ; comme $p_k^a = \rho_a(k)$ nous avons $0 \leq p_k^a \leq 1$ aussi.

En ce qui concerne l'induction, supposons que $a \leq j < n$, $k \in \mathrm{Supp}(\rho_j)$, $0 \leq p_k^j \leq 1$ et $\sum_{m=a}^{j-1} \ell_m^j p_k^m \leq \rho_j(k)$. Nous avons alors

$$\sum_{m=a}^{n-1} \ell_m^n p_k^m \leq \sum_{m=\eta(\rho,k)}^{n-1} \ell_m^n p_k^m \leq \sum_{m=\eta(\rho,k)}^{n-1} \ell_m^n \leq \rho_n(k).$$

La première inégalité est vérifiée car $p_k^m = 0$ pour $m < \eta(\rho,k)$ puisque $\mathcal{A}$ respecte le schéma de récurrence ; la seconde inégalité est due à une des hypothèses d'induction ; la dernière inégalité fait partie des hypothèses du lemme. Nous obtenons de plus $p_k^n \geq 0$ en utilisant à nouveau le schéma de récurrence.

Par conséquent, $\mathcal{A}$ est bien défini, et donc simule ρ. □

5. Plus précisément, on ne devrait pas parler d'algorithmes *stricto sensu* lorsque les coefficients p_k^m sont négatifs. Le point clé est de respecter le schéma en deux étapes, le fait d'être *bien défini* implique la simulation de ρ. Si on veut être précis, il faudrait préciser ce que fait l'« algorithme » lorsque les probabilités p_k^m qu'il *calcule* se trouvent être négatives : par exemple, il renvoie « échec » ou un élément de ρ_a, peu importe.

Simulation des familles à supports finis

Algorithme 26 Algorithme de simulation pour ρ

$D \leftarrow \emptyset$; $e \leftarrow 0$
tant que $|D| \neq a$ **faire**
  répéter
    $d \leftarrow$ `draw()`
  jusqu'à $d \notin D$
  $D \leftarrow D \cup \{d\}$
répéter
  $d \leftarrow$ `draw()`
  si $d \in D$ **alors**
    $e \leftarrow e + 1$
  sinon
    $D \leftarrow D + d$; $e \leftarrow 0$
jusqu'à $e \geq \alpha(|D|)$
$m \leftarrow |D|$
retourner k avec probabilité $p_k^m = \left(\rho_m(k) - \sum_{j=a}^{m-1} \ell_j^m p_k^j\right) / \ell_m^m$
▷ α est donné par la formule 5.4 et ℓ_j^m par la formule 5.5.

Dans ce paragraphe, nous prouvons que, dès lors que chaque probabilité $\rho_n(k)$ d'une famille à supports finis (pour k, un élément de son support) est ultimement supérieure à une fonction calculable et décroissante polynomialement en n, alors la famille dans son ensemble est simulable.

Théorème 5.11. *Si pour une famille $\rho = (\rho_n)_{n \geq a}$ à supports finis, il existe une fonction calculable f : Supp$(\rho) \to \mathbb{N}$ telle que pour tout $k \in$ Supp(ρ), pour tout $n \geq \max\{2, \eta(\rho, k)\}$, $\rho_n(k) \geq 1/n^{f(k)}$, alors ρ est simulable.*

Avant de passer à la preuve, nous allons décrire l'algorithme de simulation que nous proposons (voir algorithme 26). On rappelle que sous nos hypothèses, il existe une certaine fonction calculable f, telle que la probabilité $\rho_n(k)$ est bornée inférieurement par $n^{-f(k)}$, et ce pour tout élément k du support de ρ, à partir de la première apparition de k dans ce support (ou à partir de 2 si $k \in \text{Supp}(\rho_1)$).

Nous allons simuler ρ par un algorithme en deux étapes, de la manière suivante :

1. Nous tirons des éléments uniformes sur V jusqu'à en accumuler a différents.
2. Nous générons ensuite des éléments uniformes sur V et comptons au fur et à mesure le nombre d'éléments *vieux* d'affilée (longueur de la série sans nouveaux éléments). Lorsque $\alpha(m)$ vieux éléments sont vus d'affilée (où m est le nombre d'éléments vus jusqu'à présent), l'algorithme s'arrête et retourne k en se basant sur le schéma de récurrence.

Dans cette description, un élément est considéré comme vieux s'il avait déjà été tiré par le passé. La borne sur le nombre d'éléments vieux consécutifs est calculée ainsi :

$$\alpha(m) = \left\lceil (\gamma(m)+1)\frac{\log(m+1)}{\log(m+1)-\log(m)} \right\rceil \tag{5.4}$$

avec

$$\gamma(m) = \max\{f(j) \mid j \in \mathrm{Supp}(\rho_m)\}$$

où γ est bien défini car par hypothèse $\mathrm{Supp}(\rho_m)$ est toujours fini. Notons que $\gamma(m)$ et $\alpha(m)$ sont elles-mêmes calculables sous nos hypothèses sur la famille ρ.

Pour $a \leq m \leq n$, la probabilité ℓ_m^n utilisée par $\mathcal{A}$ est exactement la probabilité que $|D| = m$ à la fin de l'algorithme, en supposant que l'ensemble sous-jacent à la fonction `draw()` est de taille n. Elle est égale à la probabilité d'obtenir $\alpha(m)$ vieux éléments à la suite, multipliée par la probabilité que chacun des $m-a$ tours de boucle précédents ont tous *échoué*, *i.e.*

$$\ell_m^n = \left(\frac{m}{n}\right)^{\alpha(m)} \prod_{j=a}^{m-1} \left(1-\ell_j^n\right). \tag{5.5}$$

Une implémentation est donnée par l'algorithme 26. La preuve suivante prouve le théorème 5.11 en montrant que l'algorithme $\mathcal{A}$ décrit ci-dessus est un algorithme en deux étapes bien défini pour la famille de lois ρ.

Preuve (du théorème 5.11) Soit $n \geq a$ et $k \in \mathrm{Supp}(\rho_n)$, nous prouvons que $\sum_{m=\eta(\rho,k)}^{n-1} \ell_m^n \leq \rho_n(k)$; ainsi l'application du lemme 5.10 conclut la preuve car $\mathcal{A}$ est effectivement un algorithme en deux étapes avec probabilité de première étape ℓ_m^n et qui respecte le schéma de récurrence sur les p_k^m. Comme l'inégalité est triviale lorsque $n = a$, nous supposons dans la suite $n \geq a+1 \geq 2$.

Soient $n_k = \eta(\rho,k)$ et $n_k \leq m < n$.

Remarquons tout d'abord que comme les $(1-\ell_j^n)_{j\in[\![a,m-1]\!]}$ sont des probabilités, nous avons $\ell_m^n \leq (m/n)^{\alpha(m)}$. De plus, la function $\frac{\log(n)}{\log(n/m)} = 1 + \frac{\log(m)}{\log(n)-\log(m)}$ est décroissante car clairement $x \mapsto 1 + \frac{a}{x-b}$ est décroissante sur $]b,+\infty[$ (si $a \geq 0$).

Nous avons ainsi $\frac{\log(n)}{\log(n/m)} \leq \frac{\log(m+1)}{\log((m+1)/m)}$ puisque $n \geq m+1$, d'où $\frac{\log(n)(\gamma(m)+1)}{\log(n/m)} \leq \alpha(m)$. Alors,

$$\begin{aligned}
\left(\frac{m}{n}\right)^{\alpha(m)} &\leq \left(\frac{m}{n}\right)^{\frac{\log(n)(\gamma(m)+1)}{\log(n/m)}} \\
&\leq \left(\frac{m}{n}\right)^{\frac{-\log(n)(\gamma(m)+1)}{\log(m/n)}} \\
&\leq n^{-(\gamma(m)+1)}.
\end{aligned}$$

Donc $\ell_m^n \leq n^{-(\gamma(m)+1)} \leq n^{-(\gamma(n_k)+1)}$ puisque la fonction $\gamma(m)$ est croissante en m (à cause de la propriété d'inclusion des supports). Nous avons ainsi

$$\sum_{m=n_k}^{n-1} \ell_m^n \leq (n-n_k) n^{-(\gamma(n_k)+1)} \leq n^{-\gamma(n_k)}.$$

Par définition $f(k) \leq \gamma(n_k)$, nous avons donc $\sum_{m=n_k}^{n-1} \ell_m^n \leq 1/n^{f(k)} \leq \rho_n(k)$. □

Remarque Dans la preuve précédente, nous n'avons pas cherché à obtenir un algorithme efficace (notamment sur les différentes bornes utilisées). L'objectif était uniquement de prouver la simulabilité de familles de distributions de probabilité dans notre modèle.

D'après le théorème 5.6 et le théorème 5.11, nous avons d'ores et déjà caractérisé l'ensemble des familles de lois à supports finis simulables : ce sont exactement celles qui sont simulables par un algorithme fonctionnant en deux étapes.

5.2.4 Simulation des familles à supports infinis

En ce qui concerne les familles de distributions à supports infinis, les algorithmes en deux étapes ne couvrent pas toutes les familles simulables comme ce fut le cas pour les supports finis.

Exemple 5.2. Un exemple simple de famille à support infini impossible à simuler via un algorithme en deux étapes est une géométrique Géo'$(1-1/n) = (\rho_n)_{n\geq 2}$ de paramètre $1-1/n$, *i.e.* $\rho_n(k) = 1/n^k(1-1/n)$.

Cette distribution est facilement simulable : il suffit de compter de nombre de tirages identiques à un certain élément b_0, préalablement identifié, avant de voir un élément différent de b_0.

En notant ℓ_m^n la probabilité de première étape d'un algorithme en deux étapes pour Géo'$(1-1/n)$, nous avons, pour $k \neq 1$, $p_k^2 = 1/2^{k+1}$ et

$$p_k^3 = (2/3^{k+1} - \ell_2^3/2^{k+1})/\ell_3^3 = (2^{k+2} - 3^{k+1}\ell_2^3)/(6^{k+1}\ell_3^3)$$

Donc pour k suffisamment grand $p_k^3 < 0$, et l'algorithme n'est pas bien défini. ■

D'après l'exemple précédent, les algorithmes en deux étapes ne capturent pas la simulabilité pour les familles à supports infinis car des distributions simulables ne le sont pas par des algorithmes en deux étapes.

Naturellement, les mêmes arguments que l'exemple 5.2 sont étendus à des familles de lois arbitraires ρ tant que $\lim_{k\to\infty} \rho_n(k)/\rho_{n-1}(k) = 0$ pour un certain n. En effet, en supposant l'algorithme bien défini jusqu'au rang $n-1$, et $|\mathrm{Supp}(\rho_{n-1})| = \infty$, nous obtenons

$$p_k^n = \frac{\rho_n(k) - \sum_{m=a}^{n-1} \ell_m^n p_k^m}{\ell_n^n} = \frac{\rho_n(k) - \sum_{m=a}^{n-1} \frac{\ell_m^n}{\ell_m^{n-1}} \ell_m^{n-1} p_k^m}{\ell_n^n} \leq \frac{\rho_n(k) - c_n \cdot \rho_{n-1}(k)}{\ell_n^n} < 0$$

pour k suffisamment grand ; ici, $c_n = \min\{\ell_m^n/\ell_m^{n-1} \mid a \leq m \leq n-1\}$.

Néanmoins, en itérant l'exécution d'un algorithme en deux étapes simulant une version bornée d'une famille à supports infinis [6], nous pouvons simuler une telle famille. Ci-après est le dernier résultat de cette section, qui utilise les même critères de simulabilité que le théorème précédent.

Théorème 5.12. *Si pour une famille $\rho = (\rho_n)_{n\geq a}$ à supports infinis, il existe une fonction calculable f : Supp$(\rho) \to \mathbb{N}$ telle que pour tout $k \in$ Supp(ρ), pour tout $n \geq \max\{2,\eta(\rho,k)\}$, $\rho_n(k) \geq 1/n^{f(k)}$, alors ρ est simulable.*

Preuve Soit $\rho = (\rho_n)_{n\geq a}$ une famille de distributions de probabilité à supports infinis. Sans pertes de généralité nous supposerons ici que le support de ρ est l'ensemble des entiers non nuls, *i.e.* Supp$(\rho) = \mathbb{N} \setminus \{0\}$.

Pour $j \geq 1$, soit $\rho^j = (\rho_n^j)_{n\geq a}$ une famille de lois valide basée sur ρ mais possédant un support de taille $j+1$. Nous définissons pour $1 \leq k \leq j$, $\rho_n^j(k) = \rho_n(k)$ et $\rho_n^j(j+1) = 1 - \sum_{k=1}^{j} \rho_n(k)$.

Remarquons que pour $k \leq j$, $\rho_n^j(k) \geq 1/n^{f(k)}$ et $\rho_n^j(j+1) \geq \rho_n(j+1) \geq 1/n^{f(j+1)}$, donc les critères de simulabilité pour les familles à supports finis s'appliquent à ρ^j et le théorème 5.11 nous indique qu'il existe un certain algorithme $\mathcal{A}_j$ la simulant. Nous définissons à présent un algorithme $\mathcal{A}$, basé sur les $\mathcal{A}_j$, dans le but de simuler ρ, qui se comporte comme suit :

1. On pose $j \leftarrow 1$ au début.
2. On exécute $\mathcal{A}_j$ qui retourne une certaine valeur $X \sim \rho^j$, puis un des trois cas suivants s'applique :
   — Si $X < j$, on recommence l'étape 2.
   — Si $X = j$, on retourne X.
   — Si $X = j+1$, on recommence l'étape 2 mais avec $j \leftarrow j+1$.

Nous notons A_i l'évènement {le i-ème tour de boucle de $\mathcal{A}$ est exécuté}, *i.e.* l'algorithme $\mathcal{A}_i$ est exécuté au moins une fois lors de l'exécution de l'algorithme $\mathcal{A}$. On notera qu'un entier k en particulier ne peut être renvoyé qu'à un seul tour de boucle, le $(k+1)$-ème pour être précis.

Nous prouvons maintenant par induction sur i, que $\mathbb{P}_n(A_i) = 1 - \sum_{k=1}^{i-1} \rho_n(k) = \rho^{i-1}(i)$ et que nous avons $\mathbb{P}_n(\mathcal{A}, i) = \rho_n(i)$.

Pour le cas de base $i = 1$, cela revient à $\mathbb{P}(A_1) = 1$ (qui est trivial) et $\mathbb{P}_n(\mathcal{A}, 1) = \rho_n(1)$. Notons que le premier *sous-algorithme* exécuté par $\mathcal{A}$ est $\mathcal{A}_1$. Par définition de $\mathcal{A}_1$, $\mathbb{P}_n(\mathcal{A}_1, 1) = \rho_n^j(1) = \rho_n(1)$ et $\mathbb{P}_n(\mathcal{A}_1, 2) = \rho_n^j(2) = 1 - \rho_n(1)$. De plus, comme nous l'avons déjà fait remarquer, 1 peut seulement être retourné au premier tour de boucle. Donc $\mathbb{P}_n(\mathcal{A}, 1) = \mathbb{P}_n(\mathcal{A}_1,1) = \rho_n(1)$.

Prouvons maintenant l'étape d'induction pour $i > 1$. Pour tout i, soit $X_i \sim \rho^i$.

6. Par la propriété d'inclusion des supports, ρ_n a (potentiellement) un support fini pour les premières valeurs de n, et à partir d'un certain $n' \geq a$, le support de $\rho_{n'}$ est infini.

Nous avons donc, en examinant les trois cas du tour i,

$$\begin{aligned}\mathbb{P}_n(A_i) &= \mathbb{P}_n(A_{i-1}) \cdot \mathbb{P}(X_{i-1} = i \mid X_{i-1} \geq i-1) \\ &= \rho^{i-2}(i-1)\frac{\rho^{i-1}(i)}{1-\sum_{k=1}^{i-2}\rho_n(k)} \\ &= \rho^{i-1}(i)\end{aligned}$$

où nous utilisons notre hypothèse d'induction sur A_{i-1}. Calculons $\mathbb{P}_n(\mathcal{A}, i)$,

$$\begin{aligned}\mathbb{P}_n(\mathcal{A},i) &= \mathbb{P}_n(A_i) \cdot \mathbb{P}(X_i = i \mid X_i \geq i) \\ &= \rho^{i-1}(i)\frac{\rho_n(i)}{1-\sum_{k=1}^{i-1}\rho_n(k)} \\ &= \rho_n(i)\end{aligned}$$

Donc $\mathcal{A}$ simule bien ρ.

□

Remarque Dans la preuve précédente, on pourrait considérer d'utiliser des incréments plus importants que 1, comme par exemple en considérant ρ^{2^i} au i-ème tour de boucle. La preuve sera de même nature dans ce cas aussi.

Les théorèmes 5.12 et 5.11 prouvent le théorème énoncé au début de cette section.

5.3 Maintenance de graphes et simulabilité de lois

Nous terminons ce chapitre par quelques discussions sur des problématiques liées à la maintenance de graphes aléatoires sans connaître la taille et à la simulation en aveugle.

5.3.1 Un exemple où la suppression est impossible

Nous avons montré que dans la cas des graphes d'Erdős–Rényi avec p fixe, la maintenance du modèle était impossible car il n'existait pas d'algorithme d'insertion. Il se trouve que cette famille de graphes aléatoires possède un algorithme de suppression très simple et qui n'utilise aucune information globale : retirer les arêtes incidentes au sommet à supprimer.

Nous montrons par un exemple qu'il est à tout à fait possible que l'insertion d'un sommet (en préservant la loi) soit une opération réalisable, mais que la suppression ne soit pas possible.

Considérons la distribution suivante sur V, avec $n = |V| \geq 3$: G est un graphe aléatoire à n sommets contenant soit

— une arête choisie uniformément avec probabilité $1/2^n$;

— un triangle (trois sommets connectés) choisi uniformément, avec probabilité $1 - 1/2^n$.

Pour insérer un sommet u dans une telle distribution sur V, il suffit de tirer des sommets avec `RV` jusqu'à récupérer un sommet v de degré non-nul. Si v est de degré 1, on supprime l'arête et ajoute une nouvelle arête uniforme dans $V + u$ avec probabilité $1/2$, et avec probabilité complémentaire on choisit un sous-ensemble de taille 3 uniforme dans $V + u$, et on ajoute les 3 arêtes entre ces trois sommets pour former un triangle. Si v est de degré 2, on supprime le triangle qu'il forme avec ses voisins et choisit un nouveau triangle uniforme sur $V + u$ comme décrit précédemment. Nous conservons ainsi la distribution visée en suivant cette procédure.

Supposons maintenant qu'il existe un algorithme de suppression $\mathcal{A}$ qui préserve la distribution décrite. Nous construisons l'algorithme de simulation pour une loi de Bernoulli, supposant $|V| \geq 3$, suivant :

— tirer des éléments uniformes de V jusqu'à obtenir trois éléments a,b,c ;
— simuler $\mathcal{A}(g_{a,b,c}, a)$ sur le graphe $g_{a,b,c}$ où a, b et c forment un triangle et chaque sommet obtenu par la primitive `RV` en dehors de $\{a,b,c\}$ est isolé ;
— si lorsque $\mathcal{A}$ termine, le graphe en sortie ne contient qu'une arête, alors renvoyer 1 ; sinon renvoyer 0.

Nous entendons ici par « simuler » $\mathcal{A}$ la procédure suivante : quand l'algorithme $\mathcal{A}$ demande d'appeler `RV`, un élément uniforme est obtenu via `draw()` ; quand l'algorithme demande le voisinage d'un sommet v, $\emptyset$ est renvoyé lorsque $v \notin \{a,b,c\}$ et sinon $\{a,b,c\} - v$ est renvoyé.

Puisque le graphe obtenu après suppression ne peut être issu d'un graphe constitué d'une unique arête au maximum avec probabilité $1/2^n$, la probabilité d'obtenir une arête à partir de tous les *graphes triangles* est au moins de $1/2^{n-1} - 1/2^n = 1/2^n$; de surcroît, cette probabilité est au plus de $1/2^{n-1}$ afin de préserver la loi, *i.e.*

$$\frac{1}{2^n} \leq \sum_{g_{u,v,w}} \mathbb{P}(\mathcal{A}(g_{u,v,w}, a) = \bullet\!\!-\!\!\bullet \mid G = g_{u,v,w}) \cdot \mathbb{P}(G = g_{u,v,w}) \leq \frac{1}{2^n - 1}.$$

Donc, il doit exister au moins un graphe triangle $g_{u,v,w}$, contribuant à hauteur de $1/((2^n - 1)\binom{n}{3})$ à la probabilité d'obtenir une arête, *i.e.*

$$\mathbb{P}(\mathcal{A}(g_{u,v,w}, a) = \bullet\!\!-\!\!\bullet \mid G = g_{u,v,w}) \geq \frac{1}{(2^n - 1)\binom{n}{3}}.$$

Sous l'hypothèse que l'algorithme $\mathcal{A}$ préserve bien la distribution décrite, la probabilité que le graphe $\mathcal{A}(g_{a,b,c}, a)$ ne soit constitué d'une unique arête doit être d'au moins $1/((2^n - 1)\binom{n}{3}^2) > 0$; ceci provient de la probabilité de choisir le triplet u,v,w à l'étape 1.

L'algorithme décrit simule donc une loi de Bernoulli dont le paramètre est compris entre $1/((2^n - 1)\binom{n}{3}^2)$ et $1/2^{n-1}$. Comme, d'après le théorème 5.2.4, une telle loi est impossible à simuler à partir uniquement de tirages d'uniformes sur un ensemble de taille n, l'algorithme $\mathcal{A}$ n'existe pas.

5.3.2 Distribution de degrés non simulable

À la section 5.1.2, nous avons vu que la distribution des graphes $G(n,p)$ était non maintenable. Or, le théorème 5.11 nous indique que dans ce cas précis la distribution des degrés, et en particulier le degré du nouveau sommet, est non simulable. Nous montrons dans cette section que les phénomènes ne sont pas directement liés, et ce même lorsque l'on impose le critère supplémentaire d'invariance par renommage présenté § 1.1.1.

Proposition 5.13. *Il existe des familles de distributions $(\mu_V)_{V\subset\Omega}$ de graphes aléatoires sur V, invariantes par renommage de sommets, et maintenables sans connaître la taille du graphe, telles que $(\delta_n)_{n\geq a}$ soit non-simulable à partir uniquement de* `RV`*, où $\delta_n \sim \delta_G(u)$ pour $V \subset \Omega$, $n = |V| \geq a$, $G \sim \mu_V$ et $u \in V$.*

La preuve de la proposition découle des propositions 5.14 et 5.15.

Afin de préserver de telles distributions, le principe est qu'il existe un sommet *particulier* v_1 dans le graphe dont le degré est distribué selon une loi non simulable, e.g. $\mathrm{Bin}(n,\, p)$. Pour obtenir une distribution de graphes aléatoires invariantes par renommage, nous rendons chaque nœud présent dans le graphe équiprobable de jouer le rôle du sommet particulier v_1. S'il est possible de *retrouver* ce sommet particulier (par exemple par un degré inhabituel), alors il est possible d'adapter des stratégies pour faire évoluer son degré au fur et à mesure des mises à jour.

Exemple 5.3. Un exemple simple de distribution semblant pouvoir être simulable selon les critères énoncés ci-dessus est la distribution d'une *étoile binomiale*. Montrons pourquoi elle ne convient pas.

Une telle étoile consiste à choisir comme centre un sommet uniforme, puis ajouter chaque branche de manière indépendante avec probabilité p. Lorsqu'il existe au moins deux arêtes, il est facile de retrouver le centre (tirer des sommets jusqu'à en obtenir un de degré ≥ 2), mais le cas où il n'y pas d'arêtes ne peut pas être détecté. Un algorithme d'insertion sera forcément défectueux soit sur un stable (graphe sans arête), soit sur un graphe possédant une unique arête.

Pour les mêmes raisons, il n'est pas possible de préserver la distribution des clients et serveurs présentés au début du chapitre de modèles (exemple 1.1). Un algorithme d'insertion pour un nouveau serveur u, ne peut pas détecter le cas où il n'existe pas de clients et donc le différencier du cas où il existe un unique client (dans ce dernier, l'arc doit être redirigé vers u avec probabilité $1/(k+1)$ ne dépendant pas du nombre de clients).

■

Dans l'exemple précédent, si on force l'étoile à posséder au moins une branche (le degré du centre ne suit plus alors exactement la loi binomiale), alors il est possible de maintenir la distribution au prix d'une complexité dans les mises à jour due à la symétrie inhérente lorsque le graphe ne contient qu'une arête.

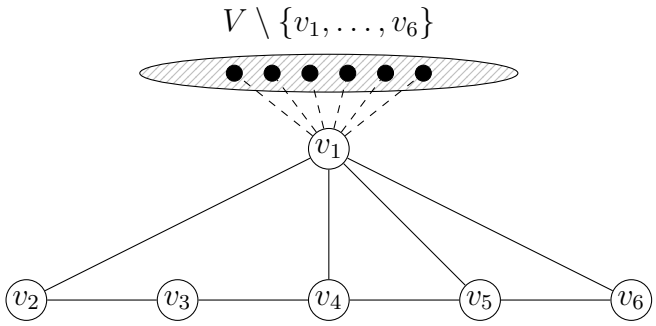


FIGURE 5.19 – Schéma des graphes distribués selon μ^p.

Pour simplifier la construction et les algorithmes de mise à jour, nous construisons une distribution en utilisant un graphe asymétrique afin d'identifier plus aisément le sommet distingué (les plus petits non-triviaux possèdent 6 sommets).

Nous considérons la distribution μ^p correspondant à une étoile binomiale de paramètre $p \in]0,1[$ *attachée* à un motif asymétrique formé par 6 sommets choisis uniformément (cf. figure 5.19).

La distribution μ^p_V est définie ainsi pour $|V| \geq 6$:

— 6 sommets distincts v_1 à v_6 sont choisis uniformément dans V, et les arêtes suivantes sont ajoutées au graphe $\{v_1,v_2\}$, $\{v_1,v_4\}$, $\{v_1,v_5\}$, $\{v_1,v_6\}$, $\{v_2,v_3\}$, $\{v_3,v_4\}$, $\{v_4,v_5\}$ et $\{v_5,v_6\}$;

— pour chaque sommet $v \in V \setminus \{v_1, v_2, v_3, v_4, v_5, v_6\}$, l'arête $\{v,v_1\}$ est ajoutée indépendamment des autres avec probabilité p.

Le figure 5.19 présente un schéma de la distribution μ^p_V : les sommets $v_1, \ldots, v_6$ forment un sous-graphe asymétrique, et le reste des sommets $V \setminus \{v_1, \ldots, v_6\}$ ont degré 0 ou 1. Sur la figure, les arêtes en pointillé sont présentes avec probabilité p de manière indépendante.

Remarque La probabilité d'obtenir un graphe $g = (V,E)$ si G est un graphe aléatoire distribué selon μ^p_V, avec $n = |V|$ et $m = |E|$, est

$$\mathbb{P}(G = g) = \frac{p^{m-8}(1-p)^{n+2-m}}{(n)_6}.$$

Nous remarquons que la distribution décrite est bien invariante par renommage des sommets. De plus la distribution δ_n du degré d'un sommet lorsque le graphe en contient n, est donnée par

$$\delta_n(j) = \begin{cases} (1-6/n)(1-p) & \text{si } j = 0 \\ (1-6/n)p & \text{si } j = 1 \\ 3/n & \text{si } j = 2 \\ 2/n & \text{si } j = 3 \\ (1/n)\binom{n-6}{j-4} p^{j-4}\,(1-p)^{n-6-(j-4)} & \text{si } j \geq 4. \end{cases}$$

Nous savons que la famille de distributions $(\delta_n)_{n\geq a}$ n'est pas simulable à partir d'un générateur d'uniformes sur $[\![n]\!]$ car la probabilité d'obtenir par exemple la valeur 4 décroit trop rapidement vers 0.

Montrons maintenant que la distribution μ^p est maintenable.

Pour supprimer un sommet u du graphe G, nous proposons l'algorithme suivant :

Algorithme 27 Sup-Bin-Star

1. Si u est de degré 0 ou 1, supprimer u et potentiellement son arête incidente.
2. Si u est de degré au moins 2, récupérer les identités des sommets v_1 à v_6 formant le motif spécial (voisins à distance au plus 3) puis sélectionner un sommet uniforme v parmi $V \setminus \{v_1, \ldots, v_6\}$; enfin retirer les arêtes incidentes à u et potentiellement l'arête incidente à v, et ajouter une arête entre v et chaque sommet de $N_G(u)$.

Proposition 5.14. Sup-Bin-Star *préserve la distribution* μ^p.

Preuve Soit G un graphe distribué selon μ^p_V, G' le graphe obtenu après suppression du sommet u et g' un graphe du support de μ^p_{V-u} à m arêtes.

Il existe 14 graphes sur V pouvant produire g' par l'algorithme : soit en conservant le même motif (2 choix selon si u est de degré 0 ou 1), soit en remplaçant un des sommets v du motif de g' par u, et en ajoutant ou non une arête incidente à v (12 possibilités au total). Dans la première situation, g' est produit de manière déterministe, alors que dans la deuxième il est produit avec probabilité $1/(n-6)$. Sommant ces différentes situations, nous obtenons la probabilité d'obtenir g' :

$$\begin{aligned}
\mathbb{P}(G' = g') &= \frac{p^{m-8}(1-p)^{n+1-m}}{(n)_6} + \frac{6}{n-6}\frac{p^{m-8}(1-p)^{n+1-m}}{(n)_6} \\
&= \frac{p^{m-8}(1-p)^{n+1-m}}{(n-1)_6}\left(\frac{n-6}{n} + \frac{6}{n}\right) \\
&= \frac{p^{m-8}(1-p)^{n+1-m}}{(n-1)_6}
\end{aligned}$$

□

Afin d'insérer un nouveau sommet u au graphe G sur V, il suffit d'enchaîner les deux étapes suivantes :

La phase de localisation de l'étape 2 est résolue dans notre modèle par des tirages successifs de sommets uniformes dans V : le sommet recherché est de degré au moins 4 ; une fois un sommet de degré non nul obtenu, il est facile de trouver celui-ci dans son voisinage (à distance au plus 2).

Algorithme 28 INS-BIN-STAR

1. Tirer un sommet v uniforme sur $V + u$: si v est dans le motif, rediriger les arêtes incidentes à v vers u, sinon continuer avec $v = u$.
2. Avec probabilité p, repérer le sommet w de type « v_1 » dans G et ajouter l'arête $\{v,w\}$; avec probabilité $1 - p$, ne rien faire.

Proposition 5.15. INS-BIN-STAR *préserve la distribution* μ^p.

Preuve Soit G un graphe distribué selon μ^p_V, G' le graphe obtenu après insertion du sommet u et g' un graphe du support de μ^p_{V+u} à m arêtes.

Si u apparaît en dehors du motif dans g', le graphe g' ne peut être obtenu qu'à partir d'un unique graphe $g = g(g')$ sur V. Dans ce cas, un sommet en dehors des sommets de type $v_1, \ldots, v_6$ de g a été sélectionné à l'étape 1 de l'algorithme d'insertion. Il existe deux cas possibles :

— si $\delta_{g'}(u) = 0$ alors le graphe g a m arêtes aussi, et

$$\mathbb{P}(G' = g') = (1 - 6/(n+1))(1-p)\mathbb{P}(G = g) = \frac{p^m(1-p)^{n+3-m}}{(n+1)_6}$$

— si $\delta_{g'}(u) = 1$ alors le graphe g a $m - 1$ arêtes, et

$$\mathbb{P}(G' = g') = (1 - 6/(n+1)) \cdot p \cdot \mathbb{P}(G = g) = \frac{p^m(1-p)^{n+3-m}}{(n+1)_6}.$$

Si u apparaît dans le motif de g', alors un sommet v du motif de G a été sélectionné à l'étape 1. Pour chaque sommet v qui n'est pas de type $v_1, \ldots, v_6$ dans g', il existe un graphe g_v sur V où le sommet u a été remplacé par v, produisant g' avec probabilité $p/(n+1)$ si $\delta_{g'}(v) = 1$ et $(1-p)/(n+1)$ si $\delta_{g'}(v) = 0$. Dans chacun des cas, nous trouvons

$$\mathbb{P}(G' = g') = (n-6)/(n+1) \cdot \frac{p^m(1-p)^{n+3-m}}{(n)_6} = \frac{p^m(1-p)^{n+3-m}}{(n+1)_6}.$$

□

Remarque La procédure d'insertion utilise en moyenne asymptotiquement moins de $1 + p^2$ tirages d'uniformes sur $V + u$, soit $\mathcal{O}(1)$ appels à `RV`.

Dans ce chapitre, nous avons caractérisé précisément les distributions dépendant d'un paramètre inconnu n et simulables à partir d'un générateur sur un ensemble de taille n.

L'exemple que nous présentons dans cette section vient contredire la conjecture (suggérée par les modèles analysés § 5.1) que la maintenabilité du graphe était directement liée à la simulabilité de la distribution des degrés.

Conclusion

Dans cette thèse, nous nous sommes intéressés au problème de maintenir des distributions données de graphes aléatoires afin de répondre à différentes problématiques posées par la modélisation de l'évolution de réseaux logiques à grande échelle. Nous avons étudié dans ce travail exploratoire la maintenabilité de plusieurs familles de graphes aléatoires pouvant modéliser des réseaux de type P2P. Pour cela, nous nous sommes placés dans un modèle local où nous n'avions pas accès à l'ensemble des sommets, mais où il était possible de tirer un sommet uniforme dans le graphe. Nous avons ensuite considérés deux sous-modèles suivant que la taille du graphe était ou non accessible lors de la procédure de mise à jour.

La table 5.5 résume les meilleures coûts obtenues en terme de nombre d'appels moyen à la primitive globale (coût de décentralisation) pour les différentes distributions considérées. Ce coût est celui des meilleurs algorithmes de préservation de lois pour les différentes distributions de graphes aléatoires étudiées dans cette thèse ; n est la taille du graphe au moment de la mise à jour. Pour ces algorithmes, la complexité en nombre moyen de pas de calcul est toujours du même ordre de grandeur que le nombre moyen d'appels à la primitive, *i.e.* $\mathcal{O}(1)$ pour toutes les distributions traitées, sauf pour les graphes d'Erdős–Rényi en connaissant la taille où elle est linéaire.

Ce travail peut être étendu dans plusieurs directions.

Il apparaît que les algorithmes utilisés pour le multigraphe de paires sont difficilement adaptables si on travaille directement sur le graphe simple sous-jacent. De même, la suppression a été réalisée de manière non décentralisée pour le modèle par attachement préférentiel considéré ici (modèle de Bollobás et Riordan). Une perspective intéressante serait d'obtenir des algorithmes décentralisés de faible complexité (sous-linéaire) pour ces deux distributions de graphes.

Toutes les distributions apparaissant dans la table 5.5 sont invariantes par renommage de sommets. Il serait intéressant d'étudier la maintenance de modèles où les identités des nœuds influencent la distribution, comme c'est généralement le cas par exemple dans des modèles de graphes géométriques. Dans ce domaine, les modèles considérés sont généralement des modèles déterministes d'un nuage de points (triangulation de Delaunay, triangulation gloutonne, graphe de disques, quadtree)

et les analyses sont effectuées en supposant un nuage de point uniforme dans un certain domaine (souvent, le carré unité). Certaines bonnes propriétés de ces structures de données sont aussi obtenues en considérant un ordre d'insertion uniforme (e.g. quadtree) sur les nœuds, et une question intéressante pourrait être de maintenir ces distributions quel que soit le nuage de points considéré (pouvant potentiellement être choisi par un adversaire), quitte à changer notre modèle de décentralisation.

Nous avons vu que le modèle $G(n,p)$ est impossible à maintenir lorsque la taille du graphe est inconnue. Ce modèle est généralement étudié dans la littérature en supposant que p est fonction de n (généralement tendant vers 0) et en faisant tendre n vers l'infini. La maintenabilité de ces graphes pour $p = f(n)$ reste ouverte et il serait déjà intéressant d'exhiber des algorithmes de mise à jour pour des cas particuliers tels que $p = c/n$, où c est constant.

Enfin, nous avons donné une caractérisation complète des familles de distributions de probabilité sur les entiers simulables à partir d'un générateur sur un ensemble inconnu de taille n. Le simulabilité de la distribution des degrés fut l'élément clé séparant la distribution non simulable sans connaître la taille du modèle $G(n,p)$ et la distribution des graphes k-sortants : les deux distributions sont des distributions binomiales mais la première n'est pas simulable et la seconde si. Nous avons montré, de manière un peu négative, que cette simulabilité ne conditionne pas dans le cas général la maintenabilité de la distribution de graphes considérée, et ce même si cette distribution est invariante par renommage des sommets. Il serait enfin intéressant d'exhiber l'existence de conditions locales contrevenant à la maintenabilité.

Distribution de graphes aléatoires	**Coût de l'insertion**		**Coût de la suppression**	
	avec taille	**sans taille**	**avec taille**	**sans taille**
Graphes d'Erdős–Rényi, p fixe	$\mathcal{O}(n)$	impossible	0	
Couplages parfaits uniformes	1		0	
Multigraphe de paires (degré m)	$m/2 + \mathcal{O}(1/n)$		0	
Attachement préférentiel (degré m)	$m + 1 + \mathcal{O}(1/n)$		non décentralisée	
Graphes k-sortants uniformes	$k + \mathcal{O}(1/n)$	$k\frac{e^2+3}{2} - 1$ $+\mathcal{O}(1/n)$	$\mathcal{O}(1/n)$	$k + \mathcal{O}(1/n)$
Graphes μ-sortants uniformes	$\lvert\mu\rvert + \mathbb{E}(\mu)$	$\mathcal{O}(\lvert\mu\rvert)$	$\mu(0)$ $+\mathcal{O}(1/n)$	$\mathbb{E}(\mu)$ $+\mathcal{O}(1/n)$

Table 5.5 – Coût de décentralisation des différentes distributions de graphes aléatoires étudiées dans cette thèse.

Annexe A

Simulations de graphes k-sortants

Nous présentons ici un ensemble complémentaire de figures obtenues à la suite de simulations sur les graphes k-sortants uniformes, venant s'ajouter à ceux présentés au cours du chapitre 3.

La totalité des tests a été réalisée à partir de programmes C (et script *gnuplot* pour les figures) disponibles sur :

`http://www.labri.fr/perso/rduvigna/code.html`.

A.1 Degré maximum et incidence du paramètre k

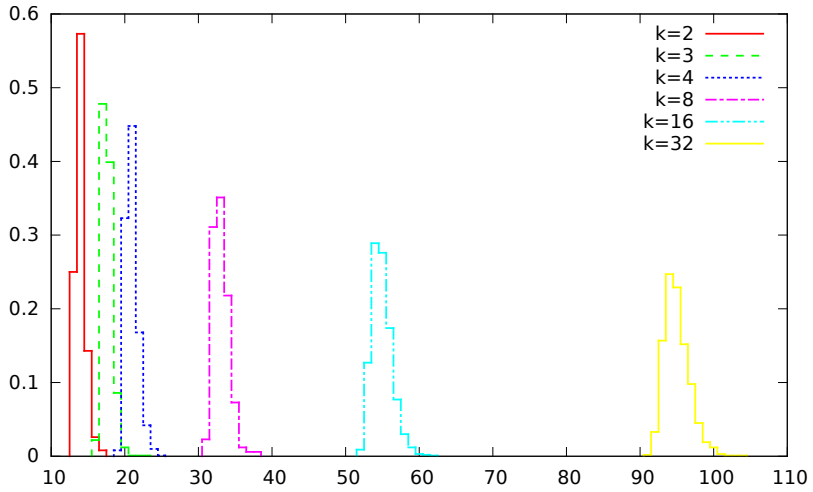


Figure A.20 – Distribution du degré maximum en fonction de k.

Nous avons déjà évoqué l'incidence du paramètre k sur plusieurs caractéristiques des graphes k-sortants (cf. discussion section 3.1.2, figure 3.6 et table 3.2), dont le degré maximum du graphe est fortement impacté tout en restant relativement faible comparé au degré moyen de $2k$.

Afin de mieux se rendre compte de l'évolution du degré maximum en fonction de k, la figure A.20 présentent le degré maximum observé lors de 10^3 simulations de graphes G_n^k indépendants avec $n = 10^6$.

A.2 Modifications de distances

Nous présentons ici trois jeux complémentaires de tests réalisés dans le même protocole que ceux présentés par la figure 3.12.

Les trois figures sont les suivantes :

— **Figure A.21** : distribution de la somme des modifications de distances après une suppression (fonction escalier), classées selon le degré entrant (sous forme d'histogrammes) lors de 10^5 simulations de graphes G_n^2 indépendants pour $n = 10^3$ (colonne de gauche) et $2{,}5 \cdot 10^4$ simulations de graphes G_n^2 indépendants pour $n = 10^4$ (colonne de droite) ; la taille des classes est de 50 (10^3 sommets) et 750 (10^4 sommets).

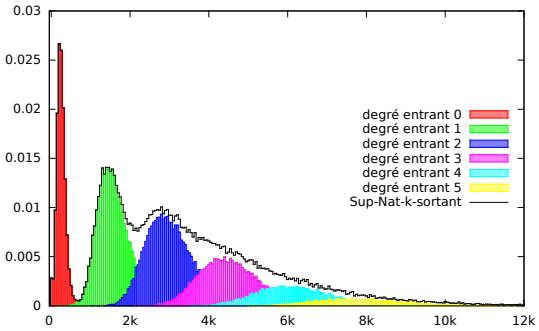
— **Figure A.22** : distribution du nombre de distances modifiées après une insertion (fonction escalier), classées selon le degré entrant (histogrammes) lors de 10^4 simulations de graphes G_n^2 indépendants pour $n = 10^3$ (colonne de gauche) et $n = 10^4$ (colonne de droite) ; la taille des classes est de 40 (10^3 sommets) et 400 (10^4 sommets).

— **Figure A.23** : distribution du nombre de distances modifiées par plus d'une unité après une suppression (fonction escalier), classées selon le degré entrant (histogrammes) lors de 10^4 simulations de graphes G_n^3 indépendants pour $n = 10^3$ (colonne de gauche) et $n = 10^4$ (colonne de droite) ; la taille des classes est de 3 (10^3 sommets) et 50 (10^4 sommets).

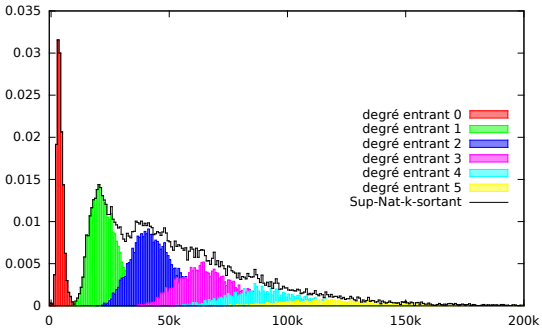
Dans les trois figures, les courbes de la dernière ligne représentent les fonctions de densité cumulative pour les trois algorithmes.

Comme énoncé lors du chapitre 3, toutes les simulations corroborent le fait que le troisième algorithme de suppression (SUP-PRED-K-SORTANT) engendre un impact sur les changements de distances significativement moins important que les deux autres algorithmes. Un phénomène analogue est observé pour le troisième algorithme d'insertion (INS-PRED-K-SORTANT) lorsqu'on le compare aux deux autres algorithmes d'insertion présentés.

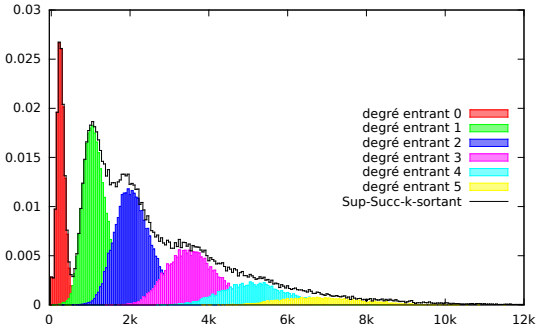
(a) Sup-Nat-k-sortant 10^3 nœuds



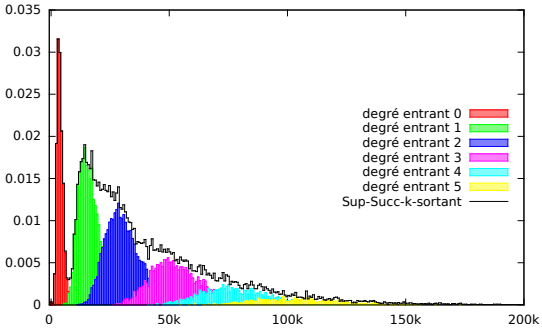
(b) Sup-Nat-k-sortant 10^4 nœuds



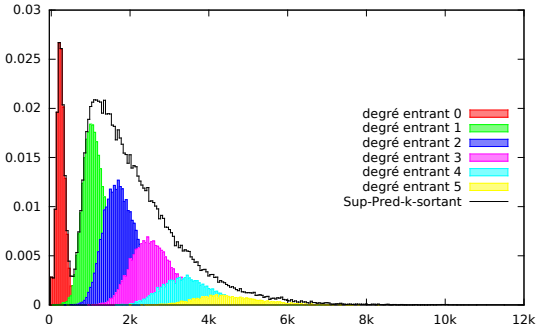
(c) Sup-Succ-k-sortant 10^3 nœuds



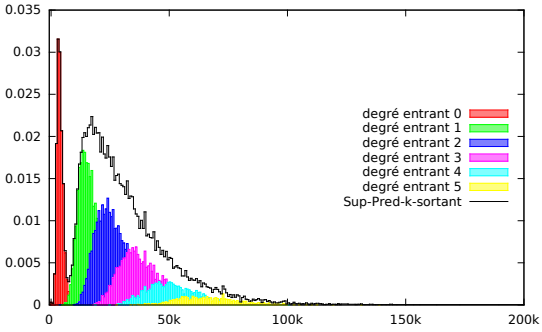
(d) Sup-Succ-k-sortant 10^4 nœuds



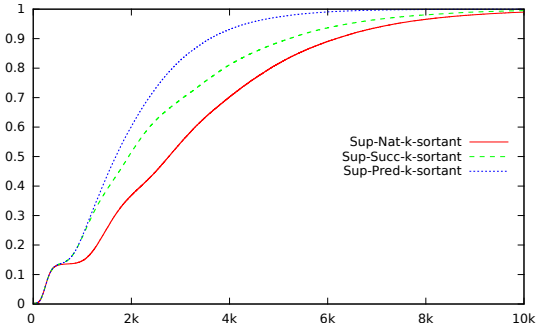
(e) Sup-Pred-k-sortant 10^3 nœuds



(f) Sup-Pred-k-sortant 10^4 nœuds



(g) Cdf 10^3 nœuds



(h) Cdf 10^4 nœuds

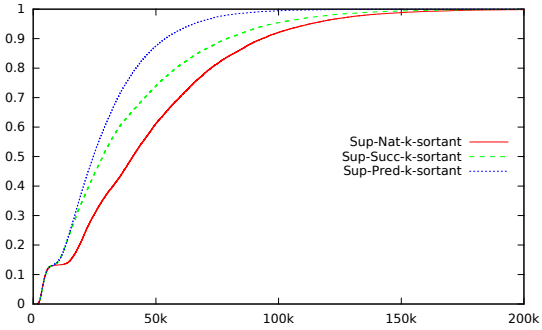
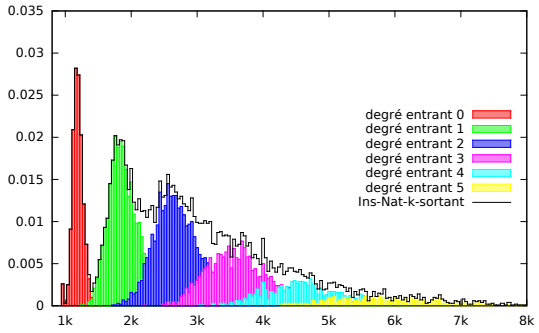
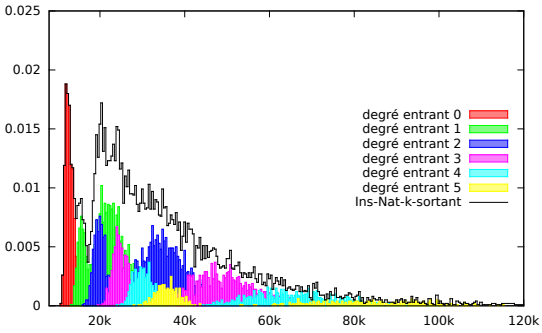


Figure A.21 – Somme des modifications de distances lors d'une suppression.

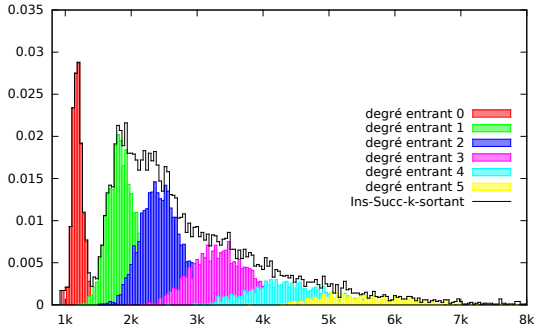
(a) INS-NAT-K-SORTANT 10^3 nœuds



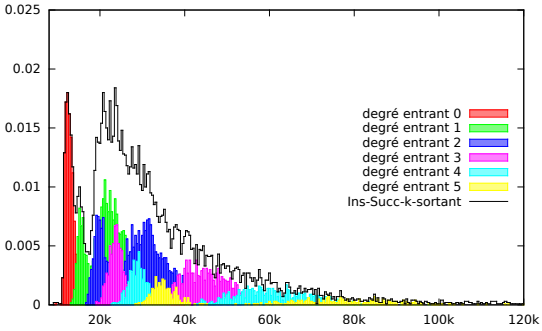
(b) INS-NAT-K-SORTANT 10^4 nœuds



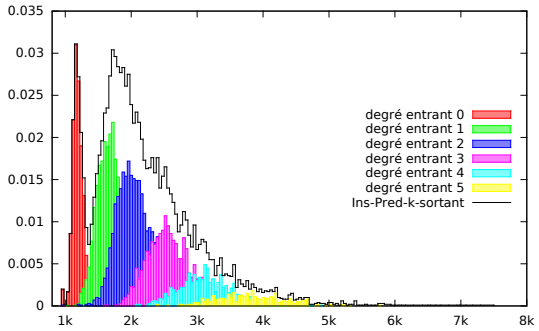
(c) INS-SUCC-K-SORTANT 10^3 nœuds



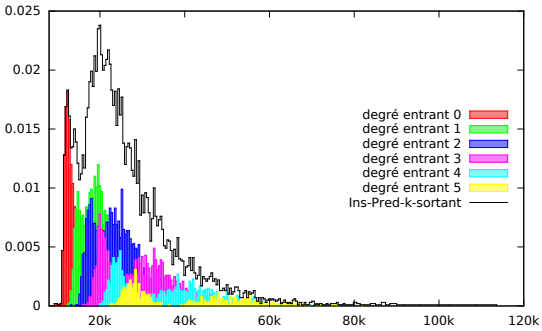
(d) INS-SUCC-K-SORTANT 10^4 nœuds



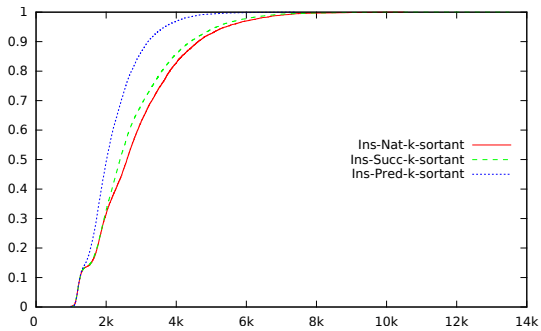
(e) INS-PRED-K-SORTANT 10^3 nœuds



(f) INS-PRED-K-SORTANT 10^4 nœuds



(g) CDF 10^3 nœuds



(h) CDF 10^4 nœuds

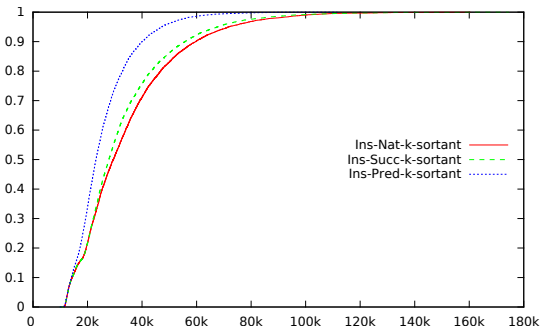
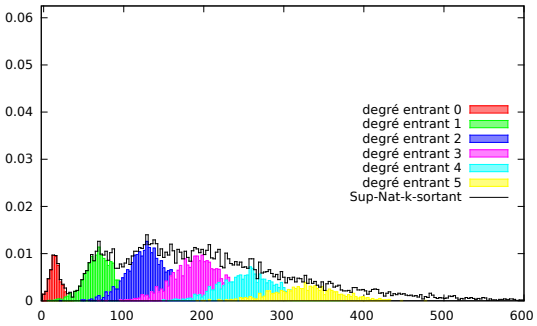
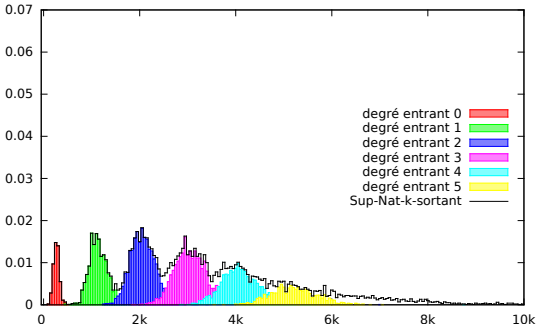


FIGURE A.22 – Nombre de modifications de distances lors d'une insertion.

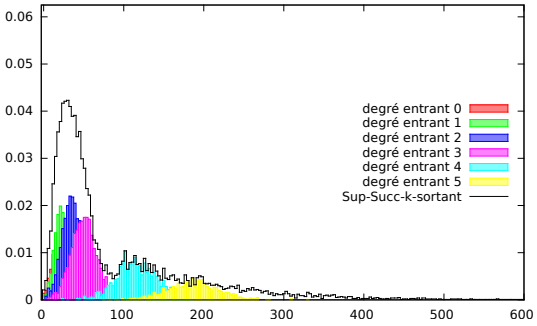
(a) Sup-Nat-k-sortant 10^3 nœuds



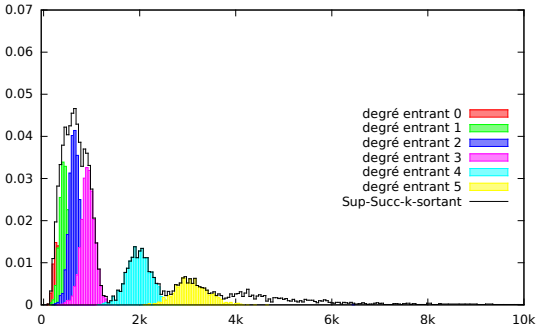
(b) Sup-Nat-k-sortant 10^4 nœuds



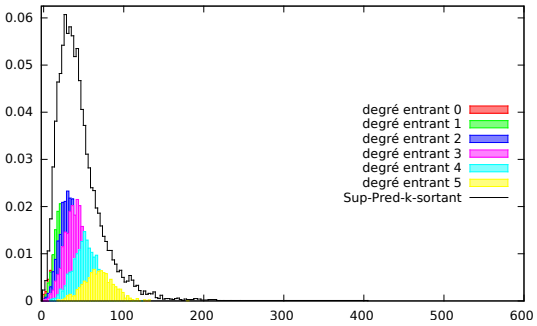
(c) Sup-Succ-k-sortant 10^3 nœuds



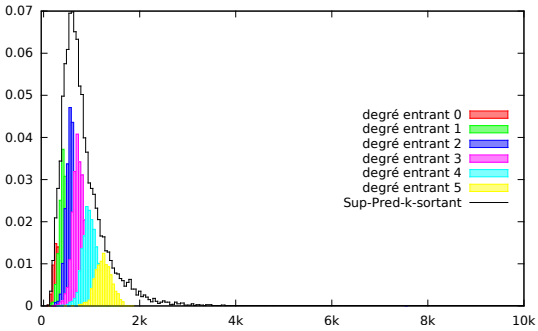
(d) Sup-Succ-k-sortant 10^4 nœuds



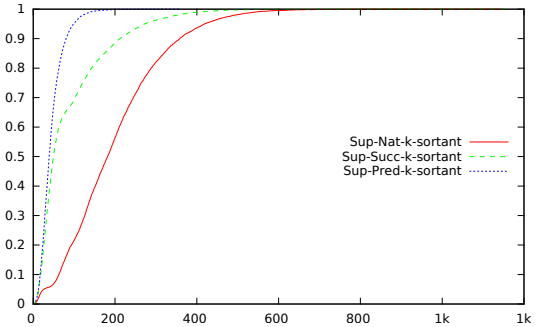
(e) Sup-Pred-k-sortant 10^3 nœuds



(f) Sup-Pred-k-sortant 10^4 nœuds



(g) Cdf 10^3 nœuds



(h) Cdf 10^4 nœuds

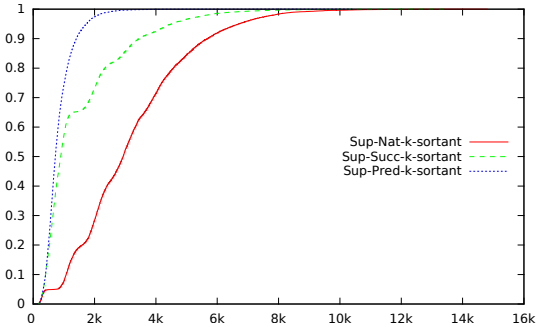


Figure A.23 – Nombre de modifications de distances par plus d'une unité, $k = 3$.

Bibliographie

[1] Karl Aberer and Manfred Hauswirth. An overview of peer-to-peer information systems. In *WDAS*, volume 14, pages 171–188, 2002.

[2] Louigi Addario-Berry, Borja Balle, and Guillem Perarnau. Diameter and stationary distribution of random r-out digraphs. *arXiv preprint arXiv :1504.06840*, 2015.

[3] Dana Angluin and Dongqu Chen. Learning a random dfa from uniform strings and state information. In *The 26th International Conference on Algorithmic Learning Theory (ALT 2015)*, 2015.

[4] Jörg Ardnt. *Generating Random Permutations*. PhD thesis, Australian National University, 2009.

[5] Sanjeev Arora and Boaz Barak. *Computational complexity : a modern approach*. Cambridge University Press, 2009.

[6] James Aspnes and Gauri Shah. Skip graphs. *ACM Transactions on Algorithms (TALG)*, 3(4) :37, 2007.

[7] Albert-Laszlo Barabasi. *Linked : How everything is connected to everything else and what it means*. Plume Editors, 2002.

[8] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439) :509–512, 1999.

[9] Frédérique Bassino, Julien David, and Cyril Nicaud. Average case analysis of moore's state minimization algorithm. *Algorithmica*, 63(1-2) :509–531, 2012.

[10] Vladimir Batagelj and Ulrik Brandes. Efficient generation of large random networks. *Physical Review E*, 71(3) :036113, 2005.

[11] Mohsen Bayati, Jeong Han Kim, and Amin Saberi. A sequential algorithm for generating random graphs. *Algorithmica*, 58(4) :860–910, 2010.

[12] Edward A Bender. The asymptotic number of non-negative integer matrices with given row and column sums. *Discrete Mathematics*, 10(2) :217–223, 1974.

[13] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9) :509–517, 1975.

[14] Lenore Blum, Mike Shub, Steve Smale, et al. On a theory of computation and complexity over the real numbers : np-completeness, recursive functions

and universal machines. *Bulletin (New Series) of the American Mathematical Society*, 21(1) :1–46, 1989.

[15] Tom Bohman and Alan Frieze. Hamilton cycles in 3-out. *Random Structures & Algorithms*, 35(4) :393–417, 2009.

[16] Béla Bollobás. A probabilistic proof of an asymptotic formula for the number of labelled regular graphs. *European Journal of Combinatorics*, 1(4) :311–316, 1980.

[17] Béla Bollobás. Degree sequences of random graphs. *Discrete Mathematics*, 33(1) :1–19, 1981.

[18] Béla Bollobás. The evolution of random graphs. *Transactions of the American Mathematical Society*, 286(1) :257–274, 1984.

[19] Béla Bollobás. *Random graphs*. Springer, 1998.

[20] Béla Bollobás and W Fernandez de la Vega. The diameter of random regular graphs. *Combinatorica*, 2(2) :125–134, 1982.

[21] Béla Bollobás and Oliver Riordan. The diameter of a scale-free random graph. *Combinatorica*, 24(1) :5–34, 2004.

[22] Béla Bollobás. *Random Graphs*. Cambridge University Press, second edition, 2001. Cambridge Books Online.

[23] Virgil Bourassa and F Holt. Swan : Small-world wide area networks. In *Proceeding of International Conference on Advances in Infrastructures (SSGRR 2003w), L'Aquila, Italy*, 2003.

[24] KM Briggs, Linlin Song, and Thomas Prellberg. A note on the distribution of the maximum of a set of poisson random variables. *arXiv preprint arXiv :0903.4373*, 2009.

[25] Karl Bringmann and Tobias Friedrich. Exact and efficient generation of geometric random variates and random graphs. In *Automata, Languages, and Programming*, pages 267–278. Springer, 2013.

[26] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W Hong. Freenet : A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies*, pages 46–66. Springer, 2001.

[27] Bram Cohen. The bittorrent protocol specification, 2008.

[28] Colin Cooper, Martin Dyer, and Catherine Greenhill. Sampling regular graphs and a peer-to-peer network. *Combinatorics, Probability and Computing*, 16(04) :557–593, 2007.

[29] Colin Cooper, Ralf Klasing, and Tomasz Radzik. A randomized algorithm for the joining protocol in dynamic distributed networks. *Theoretical Computer Science*, 406(3) :248–262, 2008.

[30] Julien David. *Génération aléatoire d'automates et analyse d'algorithmes de minimisation*. PhD thesis, Université Paris-Est, 2010.

[31] Pierre Rémond de Montmort. *Essay d'analyse sur les jeux de hazard.* Chez Claude Jombert, et Jacque Quillau, Paris, 1714. Réimprimé en 1980 par Chelsea, New York.

[32] Jacques Désarménien. Une autre interprétation du nombre de dérangements. In *Actes 8e Sém. Lothar. Combin.*, pages 11–16. IRMA, Strasbourg, 1984.

[33] Luc Devroye. *Non-Uniform Random Variate Generation.* Springer Verlag, 1986.

[34] Amalia Duch. Randomized insertion and deletion in point quad trees. In *Algorithms and Computation*, pages 415–426. Springer, 2005.

[35] Amalia Duch, Vladimir Estivill-Castro, and Conrado Martinez. *Randomized k-dimensional binary search trees.* Springer, 1998.

[36] Philippe Duchon and Romaric Duvignau. Local update algorithms for random graphs. In *LATIN 2014 : Theoretical Informatics*, pages 367–378. Springer, 2014.

[37] Philippe Duchon and Romaric Duvignau. A new generation tree for permutations, preserving the number of fixed points. In *26th International Conference on Formal Power Series and Algebraic Combinatorics (FPSAC 2014)*, pages 679–690. DMTCS Proceedings, 2014.

[38] Richard Durstenfeld. Algorithm 235 : Random permutation. *Commun. ACM*, 7(7) :420–, July 1964.

[39] P Erdős. Graph theory and probability. ii. In *Canad. J. Math.* Citeseer, 1960.

[40] Paul Erdös. Some remarks on the theory of graphs. *Bulletin of the American Mathematical Society*, 53(4) :292–294, 1947.

[41] Paul Erdős. Graph theory and probability. *Canad. J. Math*, 11 :34G38, 1959.

[42] Paul Erdős and Alfréd Rényi. On random graphs. *Publicationes Mathematicae Debrecen*, 6 :290–297, 1959.

[43] Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hungar. Acad. Sci*, 5 :17–61, 1960.

[44] Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Bull. Inst. Internat. Statist*, 38(4) :343–347, 1961.

[45] Nathan Evans, Bartlomiej Polot, and Christian Grothoff. Efficient and secure decentralized network size estimation. In *NETWORKING 2012*, pages 304–317. Springer, 2012.

[46] CT Fan, Mervin E Muller, and Ivan Rezucha. Development of sampling plans by using sequential (item by item) selection techniques and digital computers. *Journal of the American Statistical Association*, 57(298) :387–402, 1962.

[47] Trevor I. Fenner and Alan M. Frieze. On the connectivity of random m-orientable graphs and digraphs. *Combinatorica*, 2(4) :347–359, 1982.

[48] Trevor I Fenner and Alan M Frieze. On the existence of hamiltonian cycles in a class of random graphs. *Discrete mathematics*, 45(2) :301–305, 1983.

[49] R. A. Fisher and F. Yates. *Statistical Tables for Biological, Agricultural and Medical Research*. Edinburgh : Oliver and Boyd, 1938.

[50] Philippe Flajolet, Peter J Grabner, Peter Kirschenhofer, and Helmut Prodinger. On ramanujan's q-function. *Journal of Computational and Applied Mathematics*, 58(1) :103–116, 1995.

[51] Philippe Flajolet and Nasser Saheb. The complexity of generating an exponentially distributed variate. *Journal of Algorithms*, 7(4) :463–488, 1986.

[52] Mahmoud Fouz. *Randomized rumor spreading in social networks and complete graphs*. PhD thesis, Universität des Saarlandes, 2012.

[53] Justin Frankel and Tom Pepper. The gnutella project. http ://www.gnutelliums.com/.

[54] Edgar N Gilbert. Random graphs. *The Annals of Mathematical Statistics*, pages 1141–1144, 1959.

[55] Christian Grothoff, Ioana Patrascu, Krista Bennett, Tiberiu Stef, and Tzvetan Horozov. The gnet whitepaper. *Purdue University*, 2002.

[56] Andrzej Grzegorczyk. On the definitions of computable real continuous functions. *Fundamenta Mathematicae*, 44(1) :61–71, 1957.

[57] Dominique A Heger. A disquisition on the performance behavior of binary search tree data structures. *European Journal for the Informatics Professional*, 5(5) :67–75, 2004.

[58] Michaela Heyer. Randomness preserving deletions on special binary search trees. *Electronic Notes in Theoretical Computer Science*, 225 :99–113, 2009.

[59] Thomas N Hibbard. Some combinatorial properties of certain trees with applications to searching and sorting. *Journal of the ACM (JACM)*, 9(1) :13–28, 1962.

[60] Wolfgang Hörmann. The generation of binomial random variates. *Journal of statistical computation and simulation*, 46(1-2) :101–110, 1993.

[61] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees : Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663. ACM, 1997.

[62] Leo Katz. Probability of indecomposability of a random mapping function. *The Annals of Mathematical Statistics*, pages 512–517, 1955.

[63] Jeong Han Kim and Van H Vu. Generating random regular graphs. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 213–222. ACM, 2003.

[64] Gary Don Knott. *Deletion in Binary Storage Trees*. PhD thesis, Stanford University, Stanford, CA, USA, 1975. AAI7525557.

[65] Donald E Knuth. Deletions that preserve randomness. *IEEE Transactions on Software Engineering*, 5 :351–359, 1977.

[66] Donald E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.) : Seminumerical Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[67] Donald E. Knuth. *The Art of Computer Programming, Volume 3 : (2Nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

[68] Donald E Knuth and Andrew C Yao. The complexity of nonuniform random number generation. In *Proceedings of Symposium on Algorithms and Complexity*, pages 357–428. Academic Press, New York, 1976.

[69] Erwan Le Merrer, Anne-Marie Kermarrec, and Laurent Massoulie. Peer to peer size estimation in large and dynamic networks : A comparative study. In *High Performance Distributed Computing, 2006 15th IEEE International Symposium on*, pages 7–17. IEEE.

[70] David Liben-Nowell, Hari Balakrishnan, and David Karger. Analysis of the evolution of peer-to-peer systems. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 233–242. ACM, 2002.

[71] George Marsaglia, Wai Wan Tsang, Jingbo Wang, et al. Fast generation of discrete random variables. *Journal of Statistical Software*, 11(3) :1–11, 2004.

[72] Conrado Martínez, Alois Panholzer, and Helmut Prodinger. Generating random derangements. In *I. Munro, R. Sedgewick, W. Szpankowski, and D. Wagner, editors, Proc. of the 10th ACM-SIAM Workshop on Algorithm Engineering and Experiments (ALENEX) and the 5th ACM-SIAM Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*, pages 234–240. SIAM, 2008.

[73] Conrado Martínez and Salvador Roura. Randomized binary search trees. *Journal of the ACM (JACM)*, 45(2) :288–323, 1998.

[74] Yossi Matias and Yehuda Afek. Simple and efficient election algorithms for anonymous networks. In *Distributed Algorithms*, pages 183–194. Springer, 1989.

[75] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Chapman & Hall/CRC, 2010.

[76] Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal. Building low-diameter peer-to-peer networks. *Selected Areas in Communications, IEEE Journal on*, 21(6) :995–1002, 2003.

[77] David Peleg. Distributed computing. *SIAM Monographs on discrete mathematics and applications*, 5, 2000.

[78] Yuval Peres et al. Iterating von neumann's procedure for extracting random bits. *The Annals of Statistics*, 20(1) :590–597, 1992.

[79] Thomas K Philips, Donald F Towsley, and Jack K Wolf. On the diameter of a class of random graphs. *Information Theory, IEEE Transactions on*, 36(2) :285–288, 1990.

[80] Johan Pouwelse, Paweł Garbacki, Dick Epema, and Henk Sips. The bittorrent p2p file-sharing system : Measurements and analysis. In *Peer-to-Peer Systems IV*, pages 205–216. Springer, 2005.

[81] David Price. Sizing the piracy universe. *Netnames. http ://copyrightalliance.org/sites/default/files/ 2013-netnames-piracy.pdf*, 2013.

[82] William Pugh. Skip lists : a probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6) :668–676, 1990.

[83] Dongyu Qiu and Rayadurgam Srikant. Modeling and performance analysis of bittorrent-like peer-to-peer networks. In *ACM SIGCOMM Computer Communication Review*, volume 34-4, pages 367–378. ACM, 2004.

[84] Fanja Rakotondrajao. k-fixed-points-permutations. In *INTEGERS : Electronic Journal of Combinatorial Number Theory*, volume 7, 2007. A36.

[85] Robert Sedgewick and Philippe Flajolet. *An introduction to the analysis of algorithms*. Addison-Wesley, 2013.

[86] Raimund Seidel and Cecilia R Aragon. Randomized search trees. *Algorithmica*, 16(4-5) :464–497, 1996.

[87] Neil JA Sloane and others (The OEIS Foundation Inc). The on-line encyclopedia of integer sequences. http ://oeis.org/, 2015.

[88] Angelika Steger and Nicholas C Wormald. Generating random regular graphs quickly. *Combinatorics, Probability and Computing*, 8(04) :377–396, 1999.

[89] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. Chord : A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4) :149–160, 2001.

[90] John Von Neumann. Various techniques used in connection with random digits. *NBS Appl. Math. Series*, 36 :12, 1951.

[91] Chonggang Wang and Bo Li. Peer-to-peer overlay networks : A survey. *Department of Computer Science, The Hong Kong University of Science and Technology, Hong Kong*, page 9, 2003.

[92] Napster website. http ://www.napster.com/, 2003.

[93] Maciej Wojciechowski, Mihai Capotă, Johan Pouwelse, and Alexandru Iosup. Btworld : towards observing the global bittorrent file-sharing network. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 581–588. ACM, 2010.

[94] Nicholas C Wormald. The asymptotic connectivity of labelled regular graphs. *Journal of Combinatorial Theory, Series B*, 31(2) :156–167, 1981.

[95] Nicholas C Wormald. Generating random regular graphs. *Journal of Algorithms*, 5(2) :247–280, 1984.

[96] Nicholas C Wormald. Models of random regular graphs. *London Mathematical Society Lecture Note Series*, pages 239–298, 1999.

[97] Osman Yağan and Armand M Makowski. Connectivity results for sensor networks under a random pairwise key predistribution scheme. In *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, pages 1797–1801. IEEE, 2012.

Table des figures

Liste des algorithmes