# THÈSE DE DOCTORAT

de l'Université de recherche Paris Sciences Lettres
PSL Research University

## Préparée à l'École Normale Supérieure

# Simulation parfaite de réseaux fermés de files d'attente et génération aléatoire de structures combinatoires

*Perfect sampling of closed queueing networks and random generation of combinatorial objects*

**École doctorale n°386**
SCIENCES MATHÉMATIQUES DE PARIS CENTRE

**Spécialité** Informatique

**COMPOSITION DU JURY :**

Mme. Simonetta Balsamo
Università Ca' Foscari
Rapporteure

M. Alain Jean-Marie
Inria Sophia-Antipolis Méditerranée
Rapporteur

Mme. Anne Bouillard
École normale supérieure, Nokia
Membre du Jury

Mme. Ana Bušić
Inria de Paris
Membre du Jury

M. Thomas Fernique
CNRS, LIPN
Membre du Jury

M. Laurent Massoulié
Microsoft Research, Inria Saclay
Membre du Jury

Mme. Claire Mathieu
CNRS, École normale supérieure
Membre du Jury

**Soutenue par Christelle Rovetta**
**le 20 juin 2017**

Dirigée par **Anne Bouillard**
et co-dirigée par **Ana Bušić**

ENS ÉCOLE NORMALE SUPÉRIEURE | PSL RESEARCH UNIVERSITY PARIS

# Table des matières

# Introduction

La génération aléatoire d'objets combinatoires est un problème important qui se pose dans de nombreux domaines (physique, réseaux de communications, géométrie stochastique, informatique théorique, biologie, etc.). Les raisons pour lesquelles on souhaite obtenir des échantillons sont multiples : on veut par exemple évaluer les performances d'un système, trouver la réponse optimale à un problème, tester une hypothèse. Dans cette thèse, je me suis intéressée à deux techniques de génération aléatoire utilisant des outils mathématiques différents : la simulation parfaite des chaînes de Markov et la méthode de Boltzmann issue de la combinatoire analytique.

## Chaîne de Markov et simulation parfaite

Une chaîne de Markov est un processus stochastique à temps discret : à chaque instant, la chaîne prend une valeur aléatoire conditionnellement à la valeur qu'elle possédait à l'instant d'avant. Les chaînes de Markov ergodiques possèdent une *distribution stationnaire*. Cette dernière donne la limite de la fréquence d'apparition de chaque état pour une la chaîne évoluant indéfiniment. Ainsi la probabilité d'apparition pour un temps fixé d'un état après un temps d'exécution de la chaîne "assez grand" est donné par la distribution stationnaire.

Les chaînes de Markov sont utilisées notamment par l'algorithme *PageRank* de Google [48], initialement conçu pour réaliser le classement des sites les plus populaires. Le comportement d'un internaute qui navigue de lien en lien sur internet est modélisé par une chaîne de Markov. Par exemple, une chaîne ayant pour valeur : $X_n = A$ et $X_{n+1} = B$ signifie qu'à l'instant $n$ l'internaute se trouve sur une page web $A$ puis clique sur un lien et se retrouve sur la page $B$ à l'instant $n+1$. En regardant les liens hypertextes de la page web $A$, *PageRank* détermine avec quelle probabilité un internaute visitant la page $A$ va se rendre sur la page $B$ à son prochain clic. À partir de ces informations concernant tous les liens de toutes pages, *PageRank* détermine alors le taux de visite moyen de toutes les pages web, qui correspond à la distribution stationnaire de la chaîne de Markov.

Un autre exemple plus proche des thématiques de cette thèse est la modélisation d'une file d'attente. On considère une file d'attente qui, à chaque instant $n \in \mathbb{N}$, reçoit un nombre aléatoire de clients noté $A_n$, où $A_n \in \mathbb{N}$. Ainsi le nombre de clients dans la file peut être modélisé par une chaîne de Markov définie telle que $X_0 = 0$ (la file est vide à l'instant $0$) et pour tout $n \geq 1$, $X_n = \max(X_n - 1, 0) + A_n$.

Si le nombre d'états de la chaîne n'est pas trop grand la distribution stationnaire peut être déterminée en effectuant un calcul matriciel. Dans le cas contraire, si la cardinalité de l'espace des états de la chaîne ne permet pas le calcul, une des solutions consiste alors à simuler la chaîne *suffisamment longtemps* pour retrouver la distribution stationnaire : c'est ce qu'on appelle la méthode de Monte-Carlo. Le problème du biais se pose alors : combien de temps faut-il simuler la chaîne pour obtenir des échantillons dont la distribution est assez proche de la distribution stationnaire ?

La simulation parfaite fut introduite en 1996 par Propp et Wilson dans [51] : elle a pour vocation l'échantillonnage sans biais de la distribution stationnaire d'une chaîne de Markov ergodique ayant un nombre fini d'états. Le fait qu'elle fournisse un échantillon sans biais lui vaut son qualificatif de "parfaite". L'idée de Propp et Wilson est la simulation en parallèle, et depuis le passé, de toutes les trajectoires possibles de la chaîne. La figure 0.1 illustre le fonctionnement de l'algorithme de simulation parfaite. On commence la simulation de tous les états de la chaîne au temps $-1$ et on effectue un pas de simulation. À l'instant $0$, on regarde le nombre d'états différents. Si ce nombre est supérieur à 1, c'est-à-dire s'il n'y a pas eu couplage, on recommence la simulation en partant d'encore plus loin dans le passé. On arrête la simulation quand on trouve un $n$ tel que la simulation en parallèle de toutes les trajectoires en partant du temps $-n$ aboutisse à un unique état au temps $0$. La complexité en temps de calcul et en mémoire de cet algorithme est donc intrinsèquement liée au nombre d'états. Ainsi une chaîne possédant un "trop grand" nombre d'états rend l'algorithme inutilisable en pratique.

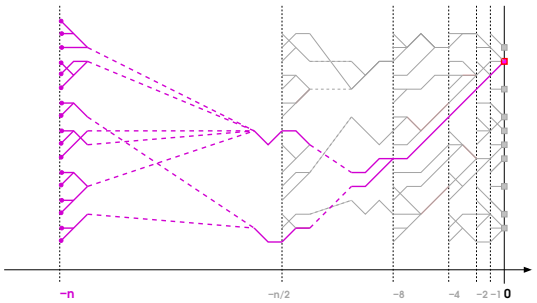



**FIGURE 0.1:** Fonctionnement de l'algorithme de Propp et Wilson.

Un des challenges lorsque l'on veut faire de l'échantillonnage par simulation parfaite réside généralement à mettre en place des stratégies permettant de ne pas avoir à simuler toutes les trajectoires [51, 39, 37, 46, 18]. Par exemple, lorsque la chaîne de Markov est monotone, il suffit de simuler un nombre restreint de trajectoires pour parvenir au résultat. Certaines de ces stratégies seront décrites dans le chapitre 1.

Dans cette thèse, on s'intéressera particulièrement à la simulation parfaite de réseaux fermés de files d'attente. L'évolution d'un réseau fermé de files d'attente est modélisée par une chaîne de Markov dont le nombre d'états est de taille exponentielle en le nombre de files.

## Réseaux de files d'attente et simulation

La théorie des files d'attente a été introduite en 1917 par Erlang dans le but de modéliser les réseaux téléphoniques. De nos jours, cette théorie intervient dans de nombreux autres domaines d'application et particulièrement dans les systèmes informatiques. La dynamique d'un réseau de files d'attente (ouvert ou fermé) est couramment représentée par une chaîne de Markov ergodique. Il y a une classe de réseaux particulièrement appréciée pour la modélisation appelée *réseaux de Jackson* [38]. Pour ce type de réseaux, l'expression de la distribution stationnaire est à *forme produit* et se calcule en temps polynomial. Tous les réseaux ne sont pas à forme produit, par exemple cela peut se produire quand on introduit de nouvelles contraintes sur le réseau telles que des files à capacité finie ou plusieurs classes de clients. Quand on ne peut pas calculer directement la distribution stationnaire, pour néanmoins obtenir des informations sur cette dernière, on a recours à des méthodes d'approximation, qui consistent à trouver une solution approchée de la distribution stationnaire [2, 44, 54], ou à la simulation stochastique [1], qui englobe : la méthode de Monte Carlo, la simulation parfaite ainsi que d'autres méthodes. En simulation, on souhaite de préférence obtenir des échantillons non-biaisés, d'où la présence de travaux alliant simulation parfaite et réseaux de files d'attente [55, 17, 16] dont nous parlerons plus en détail dans le chapitre 1.

Un réseau fermé de files d'attente est un réseau dans lequel les clients ne sont supposés ni entrer ni quitter le réseau. Le nombre d'états d'un réseau à $K$ files et $M$ clients est de l'ordre de $O(M^K)$ : ce nombre ne permet donc pas l'application naïve de l'algorithme de Propp et Wilson. De plus, la structure de l'espace des états ne permet pas l'application directe des stratégies mise en œuvre par exemple dans [51, 39, 37, 46, 18].

## Fonction génératrice et méthode de Boltzmann

En mathématiques, la combinatoire est l'étude du dénombrement d'ensembles d'objets définis par une propriété et étant indexés par une taille. On se sert de la combinatoire pour la génération aléatoire de ces objets. Par exemple, grâce à la connaissance du nombre d'arbres binaires à $n$ nœuds, l'algorithme de Rémy [52] permet la génération aléatoire et uniforme de tels arbres pour une taille $n$ donnée.

En combinatoire analytique, on associe classes combinatoires et fonctions génératrices. Le livre de référence en le domaine est "Analytics Combinatorics" de Flajolet et Sedgewick [27]. Une classe combinatoire $(\mathcal{A}, |.|)$ est un couple qui contient un ensemble d'objets $\mathcal{A}$ et une fonction $|.| : \mathcal{A} \to \mathbb{N}$ qui attribue à chaque objet une taille entière. Les coefficients $a_n$ d'une fonction génératrice $A(z) = \sum_{n\in\mathbb{N}} a_n z^n$ associée à la classe combinatoire $(\mathcal{A}, |.|)$ dénombrent les éléments de $\mathcal{A}$ de taille $n$. La méthode symbolique est une technique qui consiste à définir une classe combinatoire de manière récursive à partir de classes combinatoires plus élémentaires en utilisant des opérateurs comme le produit et l'union. Par exemple, on peut décrire $\mathcal{B}$ l'ensemble des arbres binaires de façon récursive en remarquant qu'un arbre est soit une feuille ($\mathcal{F}$) soit un nœud interne ($\mathcal{N}$) auquel deux autres arbres binaires sont accrochés (voir la figure 0.2). Ainsi, peut définir $\mathcal{B} = \mathcal{F} \cup \mathcal{N} \times \mathcal{B} \times \mathcal{B}$.

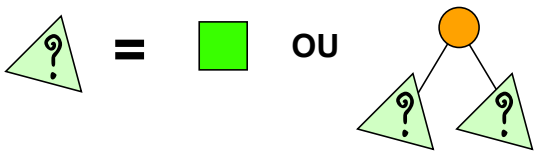



**FIGURE 0.2:** Arbre binaire.

Si on décide de compter le nombre de feuilles dans un arbre, alors la série génératrice associée à cette classe combinatoire est $B(z) = z + B(z)^2$. Si on souhaite compter les nœuds internes alors on aura $B(z) = 1 + zB(z)^2$. La méthode récursive [28] utilise la décomposition récursive des classes combinatoires afin générer uniformément des objets de mêmes tailles appartenant une même classe combinatoire. Pour générer un objet de taille $n$ récursivement, il faut calculer tous les coefficients $a_k$ pour $k \leq n$. Le problème est que le nombre d'objets combinatoires de taille $n$ peut rapidement devenir très grand. Par exemple, dans le cas des arbres binaires, $b_n \approx 4^n$. Ceci peut poser des difficultés de calculs et de stockage des coefficients nécessaires pour la génération récursive.

La méthode de Boltzmann, introduite en 2004 par Duchon, Flajolet, Louchard et Schaeffer dans [24] a pour but l'échantillonnage uniforme d'objets de même taille appartenant à une même classe combinatoire. Elle se distingue des autres méthodes de génération d'objets combinatoires ([52, 45, 28]) car elle permet la génération uniforme des objets sans avoir besoin de dénombrer l'ensemble que l'on veut échantillonner. À l'instar de la méthode récursive, elle s'appuie sur la méthode symbolique pour construire les générateurs de Boltzmann. Dans la méthode de Boltzmann, on utilise une évaluation $A(x)$ de la fonction génératrice d'une classe combinatoire afin de générer un élément de cette classe. La taille de l'objet retourné par un générateur de Boltzmann est aléatoire, cependant la variable $x$, appelée *paramètre* du générateur de Boltzmann, sert à orienter le générateur vers une taille d'objet à générer.

# Contributions et plan détaillé

La principale contribution de cette thèse est l'introduction d'une nouvelle technique de simulation parfaite pour les réseaux fermés de files d'attente. Cette technique repose sur une représentation qu'on appelle *diagramme*. Au cours de la dernière année de ma thèse, je me suis intéressée à la méthode de Boltzmann et particulièrement aux générateurs de Boltzmann pour les multi-ensembles. J'ai obtenu deux résultats en lien avec la méthode de Boltzmann ne faisant pas l'objet de publication.

## Première partie

Le chapitre 1 est un préambule nécessaire à la compréhension de la première partie. On y présente de manière indépendante la simulation parfaite et les réseaux fermés de file d'attente puis on introduit la problématique de la simulation parfaite des réseaux fermés de files d'attente.

Dans le chapitre 2, on commence par donner les propriétés des ensembles pouvant être représentés par un diagramme. Ces ensembles englobent l'espace des états pour les réseaux fermés monoclasses et multiclasses. Un diagramme est défini comme un graphe qui encode un ensemble d'éléments appartenant à un espace dit *sans mémoire*. On présente également des algorithmes relatifs aux diagrammes. Ces derniers utilisent le paradigme de programmation dynamique et permettent de dénombrer le nombre d'éléments représentés par un diagramme, d'extraire ces éléments ainsi que les générer aléatoirement.

Le chapitre 3 est consacré à la simulation parfaite des réseaux fermés de files d'attente monoclasses. On montre que la représentation par diagramme permet d'encoder tous les états d'un réseau à $K$ files et $M$ clients avec une complexité mémoire en $O(KM^2)$. On présente également un algorithme de transition agissant directement sur un diagramme. La représentation par diagramme ainsi que l'algorithme de transition permettent ainsi la réalisation de l'algorithme de Propp et Wilson. On présente ensuite une représentation encore plus compacte, en $O(KM)$, que l'on appelle *diagramme sans trou*. Les résultats de ce chapitre font l'objet des publications [10], [9] et [11].

La simulation parfaite des réseaux multiclasses est traitée dans le chapitre 4. On y généralise la technique développée dans le chapitre 3 pour la simulation parfaite des réseaux multiclasses. Les résultats de ce chapitre ont été publiés dans [12].

On clôt cette première partie avec le chapitre 5 dans lequel on s'intéresse à la simulation parfaite de réseaux possédant une ou plusieurs synchronisation(s). Pour ce faire, un nouveau type de diagramme, appelé *diagramme aggloméré* est présenté.

## Deuxième partie

La deuxième partie de cette thèse est consacrée à la méthode de Boltzmann. La première section du chapitre 6 présente cette méthode introduite en 2004 par Duchon, Flajolet, Louchard et Schaeffer dans [24]. Dans la seconde section, on montre que l'espace des états d'un réseau fermé de files d'attente (monoclasse) peut être vu comme une classe combinatoire multivaluée (possédant plusieurs fonctions taille). On présente ensuite des générateurs de Boltzmann pouvant échantillonner la distribution stationnaire d'un réseau de files d'attente à forme produit. La première section du chapitre 7 présente la méthode de Boltzmann pour les multi-ensembles et les multi-ensembles de cardinalité fixe, qui fut introduite en 2007 par Flajolet, Fusy et Pivoteau dans [26]. Ces derniers utilisent les partitions d'entiers. Dans la seconde section, on montre que l'on peut encoder toutes les partitions d'entiers par un

diagramme, on en déduit ainsi un autre générateur de Boltzmann pour les multi-ensembles de cardinalité fixe, ce dernier utilisant la représentation par diagrammes.

### Troisième partie

Le chapitre 8 constitue la dernière partie de cette thèse, il est dédié à la présentation des logiciels qui ont été développés : la *toolbox* Matlab `Clones` ([9] et [11]), le *package* Python `M – Clones` présenté dans l'article [53] ainsi qu'un autre *package* Python appelé `DiagramS`. `Clones` et `M – Clones` sont tout deux dédiés à la simulation de réseaux fermés de files d'attentes respectivement monoclasses et multiclasses. `DiagramS` est encore en phase de développement. Il implémente les algorithmes du chapitre 2 dans une classe. Les différents types de diagrammes (monoclasses, multiclasses, partitions d'entiers, permutations, etc.) héritent de cette classe.

L'article [9] qui a pour objet la *toolbox* Matlab `Clones`, a reçu le prix *Best Tool Paper Award* à la conférence ValueTools 2014.

## Bibliographie personnelle


[10] A. Bouillard, A. Bušić, and C. Rovetta. Perfect sampling for closed queueing networks. *Performance Evaluation*, 79(0) :146–159, 2014.

[9] A. Bouillard, A. Bušić, and C. Rovetta. `Clones`: CLOsed queueing Networks Exact Sampling. In *8th International Conference on Performance Evaluation Methodologies and Tools, VALUETOOLS 2014*. ICST, 2014.

[12] A. Bouillard, A. Bušić, and C. Rovetta. Perfect sampling for multiclass closed queueing networks. In *12th International Conference on Quantitative Evaluation of SysTems, QEST 2015*, 2015.

[53] C. Rovetta. `M – Clones`: Multiclass CLOsed queueing Networks Exact Sampling. In *17eme congrés annuel de la Société Française de Recherche Opérationnelle et d'Aide à la Décision ROADEF 2016*, 2016.

[13] A. Bouillard, A. Bušić, and C. Rovetta. Low complexity state space representation and algorithms for closed queueing networks exact sampling. *Performance Evaluation*, 2016.

# Première partie

Simulation parfaite

# 1 Prérequis

La dynamique d'un réseau de files d'attente est couramment représentée par une chaîne de Markov ergodique. Connaître la distribution stationnaire de cette chaîne est enjeu majeur pour qui veut étudier le comportement d'un tel réseau. Quand on ne peut pas calculer directement la distribution stationnaire, une des solutions consiste à l'échantillonner. En 1996, Propp et Wilson ont introduit l'algorithme de simulation parfaite [51]. Cette technique permet un échantillonnage sans biais de la distribution stationnaire d'une chaîne de Markov ergodique.

Les chapitres 3, 4 et 5 présentent les travaux réalisés pendant cette thèse ayant pour but la simulation parfaite de réseaux fermés de files d'attente. Le chapitre 1 présente les travaux antérieurs et les outils nécessaire à la compréhension de ces trois chapitres. On commencera par présenter de manière indépendante la simulation parfaite et les différents types de réseaux fermés de files d'attente que nous étudierons.

La simulation parfaite fait l'objet de la première section, nous verrons que la complexité en temps de calcul de l'algorithme est liée au nombre d'états de la chaîne de Markov et qui n'est donc pas effectif pour des chaînes possédant un trop gros nombre d'états. Cependant, pour certains types de chaînes, on présentera des techniques [46, 39, 37, 18, 30] qui contournent le problème du nombre d'états et permettent l'utilisation de la simulation parfaite.

La deuxième section est consacrée aux réseaux fermés de files d'attente, on présentera notamment les réseaux de Gordon et Newell [33]. Ces derniers disposent d'une formule explicite pour le calcul de la distribution stationnaire. Compte tenu de la structure de leur espace d'états et leur distribution stationnaire, ils interviendront aux chapitres 2 et 5.

On s'intéressera ensuite dans la troisième section aux travaux alliant simulation parfaite et théorie des files d'attente. On verra qu'ils utilisent les techniques contournant le problème du nombre d'états. Enfin, on montrera que les techniques que l'on aura présentées ne peuvent pas s'appliquer efficacement à la simulation parfaite de réseaux fermés de files d'attente.

## 1.1 Simulation parfaite

On commence par expliquer comment simuler une chaîne de Markov. On parlera ensuite de la méthode de Monte-Carlo qui est une technique simple et naturelle pour l'échantillonnage de la distribution stationnaire. On présentera enfin l'algorithme de simulation parfaite, ses limites et ses améliorations possibles. Pour plus de détails sur ces sujets, on pourra se reporter aux ouvrages de références de Häggström [34], Asmussen et Glynn [1] ainsi qu'aux notes de Kendall [40].

### 1.1.1 Chaîne de Markov

Considérons $\mathcal{S}$ un espace au plus dénombrable et $(\mathcal{S}, \mathcal{F}, \mathbb{P})$ un espace de probabilité composé d'un ensemble $\mathcal{S}$, d'une tribu $\mathcal{F}$ sur $\mathcal{S}$ et d'une mesure de probabilité $\mathbb{P}$. On appelle loi sur $\mathcal{S}$ une suite $(\mu_s)_{s\in\mathcal{S}}$ à valeurs dans $\mathbb{R}^+$ telle que $\sum_{s\in\mathcal{S}} \mu_s = 1$. On dit qu'une variable aléatoire $X$ à valeurs dans $\mathcal{S}$ suit la loi de $\mu$ si pour tout $s \in \mathcal{S}$, $\mathbb{P}(X = s) = \mu_s$.

**Définition 1.** *Une suite $(X_n)_{n\in\mathbb{N}}$ de variables aléatoires à valeurs dans $\mathcal{S}$ est une **chaîne de Markov** homogène de loi initiale $\mu$ et de matrice de transition $P$ si*

*(1) $X_0$ suit la loi de $\mu$,*

*(2) pour tout $n \in \mathbb{N}$ et tous $s, s', s_0, \ldots, s_{n-1} \in \mathcal{S}$, on a*

$$\mathbb{P}(X_{n+1} = s' | X_0 = s_0, \ldots, X_{n-1} = s_{n-1}, X_n = s) = \mathbb{P}(X_{n+1} = s' | X_n = s).$$

Soit $P = (p_{s,s'})_{s,s'\in\mathcal{S}}$ une matrice telle que $p_{s,s'} := \mathbb{P}(X_{n+1} = s' | X_n = s)$. Cette dernière est dite **stochastique** si les deux conditions suivantes sont vérifiées :

- pour tous $s, s' \in \mathcal{S}$, $p_{s,s'} \geq 0$,
- pour tout $s \in \mathcal{S}$, $\sum_{s'\in\mathcal{S}} p_{s,s'} = 1$.

Une loi $\pi$ sur $\mathcal{S}$ est appelée **distribution stationnaire** de la chaîne de Markov $(X_n)_{n\in\mathbb{N}}$ si pour tout $s \in \mathcal{S}$, $\sum_{t\in\mathcal{S}} \pi_t p_{t,s} = \pi_s$ (*i.e.* $\pi P = \pi$). Dans la suite de cette partie, les chaînes de Markov considérées seront toujours supposées irréductibles et apériodiques, avec un espace des états fini. Dans ces conditions, elles seront donc ergodiques et posséderont chacune une unique distribution stationnaire.

**Théorème 1** (Convergence d'une chaîne de Markov)**.** *Si $(X_n)_{n\in\mathbb{N}}$ est une chaîne de Markov ergodique de distribution stationnaire $\pi$, alors $X_n$ converge en loi vers $\pi$ indépendamment de sa loi initiale $\mu$.*

Regardons maintenant comment simuler une chaîne de Markov.

### 1.1.2 Simulation d'une chaîne de Markov

On cherche à simuler la chaîne de Markov $(X_n)_{n\in\mathbb{N}}$ à valeurs dans $\mathcal{S} := \{s_1, s_2, \ldots, s_q\}$ dont on connaît la distribution initiale $\mu$ et la matrice de transition $P$. Pour alléger les notations, on définit maintenant les coefficients de la matrice $P$ par $p_{i,j} := \mathbb{P}(X_n = s_j | X_{n-1} = s_i)$.

On appelle **suite des innovations** une suite de variables aléatoires indépendantes identiquement distribuées (i.i.d.) suivant une loi uniforme sur l'intervalle $[0, 1[$. Supposons que l'on ait accès à une telle suite $(U_n)_{n\in\mathbb{N}}$.

Pour simuler $X_0$ selon la loi de probabilité de $\mu$, on utilise la variable aléatoire $U_0$. On veut que la taille de l'intervalle tel que $X_0 = s_i$ soit égale à $\mu_i$. Comme $\sum_{i=1}^{q} \mu_i = 1$, une façon simple de procéder (qui n'est pas la seule) est de découper l'intervalle $[0, 1[$ en $q$ sous-intervalles tels que pour $i \in \{1, 2 \ldots, q\}$, le $i$-ème intervalle soit de longueur $\mu_i$. On choisit ainsi $X_0 = i$ en fonction du numéro du sous-intervalle dans lequel se trouve $U_0$ en procédant de la manière suivante :

$$X_0 = \begin{cases} s_1 & \text{si } U_0 < \mu_1, \\ \vdots & \\ s_i & \text{si } U_0 \in [\sum_{k=1}^{i-1} \mu_k, \sum_{k=1}^{i} \mu_k[, \\ \vdots & \\ s_q & \text{sinon.} \end{cases}$$

On appelle **fonction de mise à jour** une fonction $\mathrm{maj} : \mathcal{S} \times [0, 1[ \to \mathcal{S}$ telle que pour tous $s_i,\ s_j \in \mathcal{S}$ et $U$ variable aléatoire suivant une loi uniforme sur l'intervalle $[0, 1[$ on a

$\mathbb{P}(\mathrm{maj}(s_i, U) = s_j) = p_{i,j}$.

Comme $\sum_{k=1}^{q} p_{i,k} = 1$, on peut, de même que pour $X_0$, choisir la fonction de mise à jour maj telle que

$$\mathrm{maj}(s_i, u) = s_j \text{ si } u \in \left[\sum_{k=1}^{j-1} p_{i,k}, \sum_{k=1}^{j} p_{i,k}\right[.$$

Pour simuler une chaîne de Markov $(X_n)_{n\in\mathbb{N}}$, on commence alors par choisir $X_0$ en fonction de $U_0$ puis pour tout $n > 0$, on pose $X_n = \mathrm{maj}(X_{n-1}, U_n)$.

**Exemple 1.** *Soit $(X_n)_n$ une chaîne de Markov à valeurs dans $\mathcal{S} = \{1, 2, 3, 4\}$ avec $\mu = (0.70, 0.10, 0.10, 0.10)$ et*

$$P = \begin{pmatrix} 0.70 & 0.15 & 0.00 & 0.15 \\ 0.20 & 0.10 & 0.70 & 0.00 \\ 0.00 & 0.30 & 0.30 & 0.40 \\ 0.20 & 0.40 & 0.20 & 0.20 \end{pmatrix},$$

*en considérant l'initialisation et la règle de mise à jour proposées précédemment, pour $U_0 = 0.62$ on a $X_0 = 1$. Pour $U_1 = 0.98$, on a $\mathrm{maj}(1, 0.98) = 4$.*

Le choix de la fonction de mise à jour peut être différent de l'exemple présenté. Il n'a pas d'influence sur le bon déroulement de la simulation de la chaîne. On verra cependant que ce choix doit être bien réfléchi pour la simulation parfaite.

### 1.1.3 La méthode de Monte-Carlo

La méthode de Monte-Carlo permet l'échantillonnage de la distribution stationnaire d'une chaîne de Markov ergodique. Elle repose sur le théorème 1, qui dit qu'en simulant assez longtemps une chaîne de Markov ergodique, la distribution de $X_n$ se rapproche de la distribution stationnaire. Dans la méthode de Monte-Carlo, on commence donc par choisir arbitrairement un état dans l'espace des états puis on simule la chaîne jusqu'à un certain temps donné $N$ en utilisant la fonction de mise à jour maj et une suite d'innovation $(U_n)_{n\in\mathbb{N}}$ et on renvoie la valeur de $X_N$. Ceci revient à calculer :

$$\begin{cases} X_0 = s & s \text{ est choisi arbitrairement dans } \mathcal{S}. \\ X_{n+1} = \mathrm{maj}(X_n, U_{n+1}) & \text{pour tout } 0 \leq n < N. \end{cases}$$

**Exemple 2.** *Reprenons la chaîne de l'exemple 1 et simulons-la. Cette dernière est irréductible et apériodique, elle possède donc une unique distribution stationnaire. La simulation de Monte-Carlo pour $s = 1$, $U_1 = 0.62$, $U_2 = 0.98$, $U_3 = 0.38$, $U_4 = 0.35$, $U_5 = 0.08$, $U_6 = 0.5$ et $N = 6$ est représentée par la trajectoire en gras en figure 1.1.*

Une fois la simulation terminée, l'algorithme fournit un échantillon biaisé de la distribution stationnaire. Pour avoir une idée du biais, on peut étudier la variance asymptotique dans le théorème central limite pour les chaînes de Markov [43, 1]. On peut également faire une étude du temps de couplage, pour obtenir une borne supérieure du temps de mélange, on utilise les techniques sur les valeurs propres [43].

La difficulté de ces études dépend de la chaîne considérée et peut s'avérer être un problème difficile en pratique. Nous verrons dans la section suivante que l'algorithme de Propp et Wilson permet de résoudre le problème du biais ainsi que celui du choix du temps de simulation nécessaire.

### 1.1.4 L'algorithme de Propp et Wilson

#### Simuler toutes les trajectoires

Dans l'algorithme de Propp et Wilson on ne simule pas une trajectoire mais un ensemble de trajectoires, chacune correspond à un état initial de la chaîne. À partir d'une suite d'innovation $(U_n)_{n\in\mathbb{Z}}$ et de $N \in \mathbb{Z}$ fixé, on simule la chaîne de Markov $(X_n^N(s))_{n\geq N}$ qui commence au temps $N \in \mathbb{Z}$ et qui a pour valeur initiale $s$, en posant

$$\begin{cases} X_N^N(s) = s, \\ X_{n+1}^N(s) = \mathrm{maj}(X_n^N(s), U_{n+1}) \quad \text{pour } n \geq N. \end{cases}$$

**Exemple 3.** *Reprenons la matrice de l'exemple 1 et considérons les chaînes $(X_n^0(s))_{n\geq\mathbb{N}}$ pour $s \in \mathcal{S}$. La figure 1.1 représente toutes trajectoires de ces chaînes pour $n = 0, \ldots, 6$, $U_1 = 0.62$, $U_2 = 0.98$, $U_3 = 0.38$, $U_4 = 0.35$, $U_5 = 0.08$ et $U_6 = 0.5$. Le trait gras (et violet) correspond à la trajectoire $X_n^0(1)$. Les trajectoires se rejoignent au temps 2 ainsi, pour tout $n \geq 2$ on obtient $X_n^0(1) = X_n^0(2) = X_n^0(3) = X_n^0(4)$.*

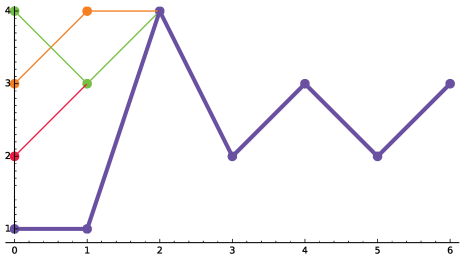



**FIGURE 1.1:** Simulation $X_n^0(s_i)$ pour $i = 1, 2, 3, 4$.

#### Simulation depuis le passé

La seconde idée de Propp et Wilson est de considérer des chaînes débutant à des temps négatifs. Le temps de simulation minimum nécessaire pour que toutes les trajectoires se rencontrent au temps $0$ est appelé **temps de couplage** (depuis le passé). Il est noté $\eta$ et est donné par la formule suivante :

$$\eta := \inf\{N \in \mathbb{N} \mid X_0^{-N}(s_1) = X_0^{-N}(s_2) = \ldots = X_0^{-N}(s_q)\}.$$

**Exemple 4.** *La figure 1.2 représente les trajectoires pour $N \in \{2, 3, 4\}$. En commençant la simulation à des temps inférieurs ou égaux à $-3$, toutes les trajectoires se rejoignent au temps $0$, ainsi le temps de couplage est ici $\eta = 3$.*

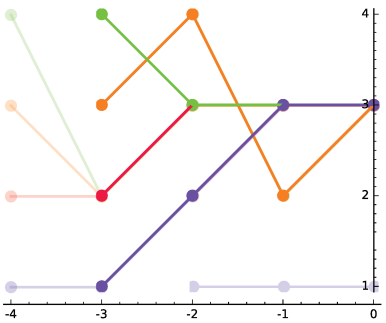



**FIGURE 1.2:** Couplage depuis le passé

Regardons de plus près la figure 1.2 et imaginons que nous fassions commencer la chaîne au temps $-5$ ou au temps $-1000$. Alors pour n'importe quelle valeur de départ $s \in \mathcal{S}$, nous obtiendrions $X_0^{-5}(s) = X_0^{-6}(s) = \cdots = X_0^{-1000}(s) = 3$.

### Algorithme

L'algorithme de Propp et Wilson est celui décrit par l'algorithme 1. On suppose que l'on dispose d'une fonction de mise à jour de la chaîne de Markov et d'une suite d'entiers positifs qui est strictement croissante $(N_k)_{k\in\mathbb{N}^*}$.

Cet algorithme réalise des simulations depuis le passé. On commence par simuler toutes les trajectoires du temps $N_1$ au temps $0$ en utilisant la fonction de mise à jour $\mathrm{maj}$. Si toutes les trajectoires ne se sont pas rejointes au temps $0$, on recommence le procédé en incrémentant $k$, et ainsi de suite jusqu'à n'obtenir qu'un seul état au temps $0$. Pour $k$ fixé, cela revient à simuler les trajectoires suivantes :

$$\begin{cases} X_{-N_k}^{-N_k}(s) = s, \\ X_{n+1}^{-N_k}(s) = \mathrm{maj}(X_n^{-N_k}(s), U_{n+1}) \quad \text{si } n \geq -N_k, \end{cases}$$

Pour ne pas avoir à simuler plusieurs fois les mêmes trajectoires du temps $-N_{k-1}$ au temps $0$, on a recours à $tab$, un tableau associatif (aussi appelé dictionnaire), qui permet de garder en mémoire le couple $(s, X_0^{-N_k}(s))$. On simule ainsi par tranches successives, de $N_k$ à $N_{k-1}$ en s'aidant du tableau associatif $tab$. Ceci est illustré par la figure 1.3. Remarquons qu'on peut choisir $N_1 = 1$, $N_2 = 2$, $N_3 = 3$, ... il s'agit alors d'une simulation depuis le passé classique. Utiliser des tranches de simulations $(N_k)_{k\in\mathbb{N}^*}$ n'est pas nécessaire dans le cas présent. Cependant on l'introduit dès maintenant car la simulation par tranche sera utilisée pour la simulation parfaite des chaînes monotones ainsi que dans les chapitres 3 et 4.

**Algorithme 1:** Algorithme de Propp et Wilson

**Données** : $\mathrm{maj}$ une fonction de mise à jour, $(N_k)_{k\in\mathbb{N}}$ une suite d'entiers positifs strictement croissante.

**début**
  **pour** ***chaque*** $s \in \mathcal{S}$ **faire**
    $tab[s] \leftarrow s$;
  $N_0 \leftarrow 0$;
  $k \leftarrow 1$;
  **tant que** $|\{tab[s] \mid s \in \mathcal{S}\}| > 1$ **faire**
    **pour** ***chaque*** $s \in \mathcal{S}$ **faire**
      $X(s) \leftarrow s$ ;
      **pour** $n = -N_k + 1, -N_k + 2, \ldots, -N_{k-1}$ **faire**
        Choisir $u$ uniformément dans $[0, 1[$ ;
        $X(s) \leftarrow \mathrm{maj}(X(s), u)$ ;
      $tab[s] \leftarrow tab[X(s)]$;
    $k \leftarrow k + 1$;
  **renvoyer** $tab[s_1]$.

La complexité mémoire de l'algorithme de Propp et Wilson est en $O(|\mathcal{S}|)$. Notons $C_{\mathrm{maj}}$ le coût d'une mise à jour pour un état. Le nombre d'itérations de la boucle **tant que** dépend du temps de couplage. La complexité en temps est donc en $O(\eta C_{\mathrm{maj}}|\mathcal{S}|)$.

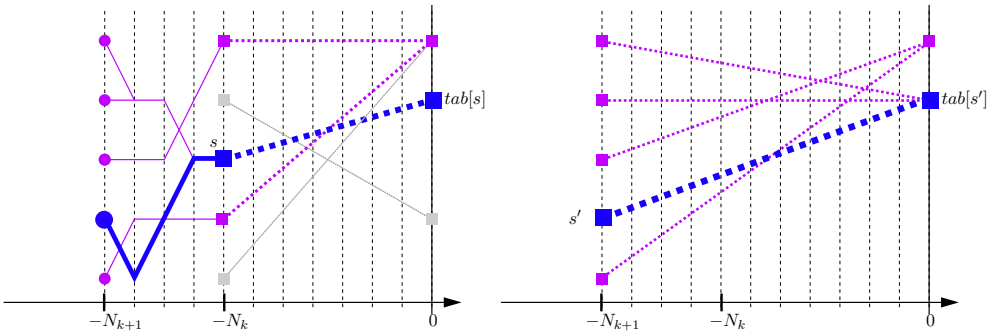



**FIGURE 1.3:** Utilisation du tableau associatif $tab$.

**Théorème 2** (Propp et Wilson [51])**.** *Supposons que l'algorithme de Propp et Wilson se termine avec probabilité 1 alors ce dernier renvoie la valeur d'une variable aléatoire distribuée selon la distribution stationnaire de la chaîne de Markov considérée.*

*Démonstration.* On reprend la démonstration de Häggström dans [34]. Soient $Y$ la valeur renvoyée par l'algorithme de Propp et Wilson, $\pi$ la distribution stationnaire de la chaîne de Markov simulée par $\mathrm{maj}$ et $s \in \mathcal{S}$ un état quelconque de la chaîne. On veut montrer que pour tout $\epsilon > 0$ on a

$$|\mathbb{P}(Y = s) - \pi_s| \leq \epsilon. \tag{1.1}$$

Fixons $\epsilon$, comme on a supposé que l'algorithme se termine avec probabilité 1, alors il existe $K \in \mathbb{N}$ tel que l'algorithme se termine avec un temps de simulation inférieur à $N_K$ avec une probabilité supérieure ou égale à $1 - \epsilon$. Fixons ce $K$. Soit $\tilde{s} \in \mathcal{S}$ un état distribué selon $\pi$ la distribution stationnaire de $(X_n)_n$, imaginons que nous simulions la chaîne suivante :

$$\begin{cases} X_{-N_K}^{-N_K}(\tilde{s}) = \tilde{s}, \\ X_{n+1}^{-N_K}(\tilde{s}) = \mathrm{maj}(X_n^{-N_K}(\tilde{s}), U_{n+1}) \quad \text{si } n \geq -N_K. \end{cases}$$

Alors en simulant $X_n^{N_k}(s)$ du temps $-N_K$ jusqu'au temps $0$, comme $\tilde{s}$ est distribué selon $\pi$, la variable aléatoire $\tilde{Y} := X_0^{-N_K}(\tilde{s})$ obtenue au temps 0 est elle aussi distribuée selon la distribution stationnaire. De plus, comme la probabilité que l'algorithme ait besoin d'un temps de simulation supérieur à $N_K$ pour s'arrêter est inférieure ou égale à $\epsilon$, on obtient ainsi :

$$\mathbb{P}(Y \neq \tilde{Y}) \leq \epsilon \tag{1.2}$$

Or

$$\begin{aligned} \mathbb{P}(Y = s) - \pi_s &= \mathbb{P}(Y = s) - \mathbb{P}(\tilde{Y} = s) \\ &= \mathbb{P}(Y = s) + \mathbb{P}(\tilde{Y} \neq s) - 1 \\ &\leq \mathbb{P}(Y = s, \tilde{Y} \neq s) \\ &\leq \mathbb{P}(Y \neq \tilde{Y}) \leq \epsilon, \end{aligned}$$

et de la même manière, on peut montrer que $\pi_s - \mathbb{P}(Y = s) \leq \epsilon$, ainsi l'inégalité (1.1) est vérifiée. □

Pour montrer que l'algorithme de Propp et Wilson se termine avec probabilité 1, il suffit de montrer que pour la fonction de mise à jour choisie, il existe une suite couplante

d'innovations. Il faut donc montrer qu'il existe $u_1, u_2, \ldots, u_L$ appartenant à l'intervalle $[0,1[$ tels que pour tout $s \in \mathcal{S}$ on obtienne la même valeur pour tous les $X_L^0(s)$ en posant :

$$\begin{cases} X_0^0(s) = s, \\ X_{n+1}^0(s) = \mathrm{maj}(X_n^0(s), u_n) \quad \text{si } n \geq 0. \end{cases}$$

Le choix de la fonction de mise à jour pour la simulation parfaite est donc primordial. L'exemple suivant illustre un mauvais choix, empêchant les trajectoires de se rejoindre.

**Exemple 5.** *Considérons une chaîne de Markov à valeurs dans* $\mathcal{S} = \{1,2,3\}$ *ayant la matrice de transition suivante :*

$$P = \begin{pmatrix} 0.00 & 0.50 & 0.50 \\ 0.50 & 0.00 & 0.50 \\ 0.50 & 0.50 & 0.00 \end{pmatrix}$$

*et pour fonction de mise à jour* $\mathrm{maj} : \mathcal{S} \times [0,1[ \to \mathcal{S}$ *donnée par*

$$\mathrm{maj}(i,u) = \begin{cases} i+1 \mod 3 & \textit{si } u \in [0,0.5[ \\ i-1 \mod 3 & \textit{sinon.} \end{cases}$$

*On s'intéresse à la simulation des chaînes* $(X_n^0(s))_{n\in\mathbb{N}}$ *pour* $s = 1,2,3$. *Pour commencer, on a* $\{X_0^0(s) \mid s \in \mathcal{S}\} = \mathcal{S}$ *et au premier pas de simulation si* $U_1 < 0.5$, *on a*

$$\{X_1^0(s) \mid s \in \mathcal{S}\} = \{X_1^0(s) + 1 \mod 3 \mid s \in \mathcal{S}\} = \mathcal{S}.$$

*De la même façon pour* $U_1 \geq 0.5$, *on a* $\{X_1^N(s) \mid s \in \mathcal{S}\} = \mathcal{S}$. *On peut alors montrer par induction que pour tout* $u \in [0,1[$, *et* $n \in \mathbb{N}$ *on a* $\{X_n^0(s) \mid s \in \mathcal{S}\} = \mathcal{S}$, *le couplage de toutes les trajectoires est donc impossible.*

*La figure 1.4 illustre la simulation (par l'avant) pour* $U = (0.87, 0.034, 0.81, 0.34, 0.95)$. *Un algorithme de simulation parfaite qui utilise cette fonction de mise à jour ne se terminera jamais tandis qu'un algorithme de Monte-Carlo utilisant cette fonction de mise à jour remplira son objectif.*

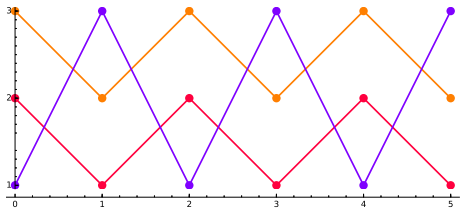



**FIGURE 1.4:** Couplage impossible.

Considérons maintenant une autre fonction de mise à jour pour la chaîne de Markov associée a la matrice $P$. Cette dernière est décrite dans le tableau 1.1. En prenant $u_1 < 0.5$ et $u_2 \geq 0.5$ on obtient pour tout $i \in \{1,2,3\}$, $\mathrm{maj}(\mathrm{maj}(i,u_1),u_2) = 3$. Ceci prouve l'existence d'une suite couplante d'événements faisant coupler la chaîne simulée en utilisant $\mathrm{maj}$. Par conséquent un algorithme de simulation parfaite qui utilise cette fonction de mise à jour se terminera avec probabilité 1.

Du moment que la chaîne est ergodique, on peut toujours trouver une fonction de mise à jour pour laquelle il existe une suite couplante d'événements. Pour montrer qu'un algorithme de simulation parfaite termine, il suffit d'exhiber une suite couplante d'événements pour la fonction de mise à jour choisie.

| $u < 0.5$ | $u \geq 0.5$ |
|---|---|
| $\mathrm{maj}(1,u) = 2$ | $\mathrm{maj}(1,u) = 3$ |
| $\mathrm{maj}(2,u) = 1$ | $\mathrm{maj}(2,u) = 3$ |
| $\mathrm{maj}(3,u) = 2$ | $\mathrm{maj}(3,u) = 1$ |

**TABLE 1.1:** Fonction de mise à jour.

Il existe des stratégies visant à améliorer la complexité en temps d'exécution et mémoire de l'algorithme de Propp et Wilson. Dans cette thèse notre intérêt se porte essentiellement celles permettant de ne pas avoir à simuler toutes les trajectoires. C'est le propos du paragraphe suivant.

### 1.1.5 Stratégies pour éviter la simulation toutes les trajectoires

On cherche généralement à échantillonner la distribution stationnaire quand l'espace des états est de cardinalité est trop grande pour pouvoir la calculer directement. Cependant une trop grande cardinalité de l'espace des états implique l'impossibilité de réaliser l'algorithme de Propp et Wilson directement car la complexité en temps d'exécution et en mémoire est au moins linéaire en la cardinalité de l'espace des états. On doit alors trouver des techniques moins coûteuses en temps de calcul et en espace mémoire qui induisent le même résultat que la simulation de toutes les trajectoires. Ces stratégies consistent généralement à considérer un représentant qui contient l'espace des états mais qui est moins coûteux à encoder et à mettre à jour. La simulation est alors réalisée directement sur ce représentant. La simulation des chaînes monotones présentée dans l'article de Propp et Wilson [51] utilise un représentant exact de l'espace des états. Les techniques de type enveloppes [46, 39, 37, 18, 30] utilisent un représentant incluant l'ensemble des états que l'on veut simuler et pouvant être plus grand que ce dernier. On présente maintenant la simulation des chaînes monotones [51] et la technique des chaînes bornantes de Huber [37]. On verra pour ces deux cas, que la stratégie choisie ne peut être appliquée que sous certaines hypothèses étroitement liées à la structure de l'espace des états et à la fonction de mise à jour de la chaîne de Markov considérée.

#### Simulation des chaînes monotone

Supposons que l'on puisse définir une relation d'ordre notée $\preceq$ (qui ne peut être que partielle) sur l'espace des états $\mathcal{S}$. On dit qu'une chaîne est *monotone* si les transitions conservent la relation d'ordre, c'est-à-dire, si :

$$s, s' \in \mathcal{S},\ s \preceq s' \implies \forall u \in [0,1[,\ \mathrm{maj}(s,u) \preceq \mathrm{maj}(s',u).$$

Supposons (pour simplifier) que $(\mathcal{S}, \preceq)$ admette deux états extrémaux notés $s_{\min}, s_{\max} \in \mathcal{S}$ tels que $s_{\min} := \min(\mathcal{S})$ et $s_{\max} := \max(\mathcal{S})$ alors le temps de couplage est donné par

$$\eta = \inf\{n \in \mathbb{N},\ X_0^{-n}(s_{\min}) = X_0^{-n}(s_{\max})\}.$$

Il n'est donc pas nécessaire de considérer toutes les trajectoires, mais seulement celles commençant par le minimum et le maximum de l'espace des états.

**Exemple 6.** *Considérons* $\mathcal{S} = \{1, 2, 3, 4\}$ *muni de la relation d'ordre* $\leq$ *(l'ordre usuel sur* $\mathbb{N}$*), la matrice de transition suivante*

$$P = \begin{pmatrix} 0.50 & 0.50 & 0.00 & 0.00 \\ 0.50 & 0.00 & 0.50 & 0.00 \\ 0.00 & 0.50 & 0.00 & 0.50 \\ 0.00 & 0.00 & 0.50 & 0.50 \end{pmatrix}$$

*et la fonction de mise à jour* maj *définie dans le paragraphe 1.1.2.*

*Si* $U_n \leq 0.5$*, alors la chaîne ayant pour valeur* $k$ *prend valeur* $\max(k-1, 1)$ *sinon elle prend pour valeur* $\min(k+1, 4)$*. À chaque transition la relation d'ordre* $\leq$ *est conservée. Seules les chaînes débutant avec les valeurs* $4$ *(le maximum) et* $1$ *(le minimum) nécessitent d'être simulées. La figure 1.5 illustre une simulation de cette chaîne.*

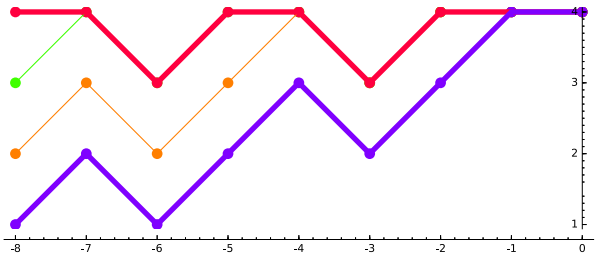



**FIGURE 1.5:** Simulation par l'arrière d'une chaîne monotone.

L'algorithme 2 est celui de Propp et Wilson dans le cas monotone. Il effectue la simulation depuis le passé des trajectoires minimale et maximale qui correspondent respectivement aux chaînes $(X_n^{-N_k}(s_{\min}))_n$ et $(X_n^{-N_k}(s_{\max}))_n$. L'algorithme s'arrête quand on trouve un $k \in \mathbb{N}$ tel que $X_0^{-N_k}(s_{\min}) = X_0^{-N_k}(s_{\max})$ et la valeur $X_0^{-N_k}(s_{\min})$ est renvoyée. Comme à chaque étape $k$, on ne simule pas toutes les trajectoires mais seulement celles issues de $s_{\min}$ et $s_{\max}$, on ne peut pas utiliser de tableau associatif pour enregistrer les tranches de simulations (comme on le faisait avec $tab$ dans l'algorithme 1). On doit alors simuler les chaînes du temps $-N_k$ au temps $0$ en utilisant la même suite d'innovation. Ceci nécessite de mémoriser cette dernière.

Le dessin de gauche de la figure 1.6 est une illustration de l'algorithme 2 pour laquelle $N_k = 2^k$ pour tout $k \in \mathbb{N}^*$. Le dessin de droite explique pourquoi l'algorithme 2 ne peut pas utiliser de tableau associatif. La complexité en temps d'exécution de l'algorithme de Propp et Wilson pour le cas monotone est en $O(\eta C_{\text{maj}})$. Ce qui correspond à une perte du facteur $|\mathcal{S}|$ en comparaison de celui dans le cas non monotone. La complexité en mémoire est en $O(\eta)$ ce qui correspond à l'enregistrement de la suite des innovations.

Propp et Wilson démontrent dans [51] que pour plus efficacité dans le cas monotone, il faut doubler le temps de simulation à chaque fois qu'on réitère dans la boucle *tant que* : choisir la suite $(N_k)_{k\in\mathbb{N}}$ telle que $N_k = 2^k$. De plus, dans [58] Wilson propose une amélioration de l'algorithme afin de ne pas avoir à enregistrer la suite des innovations.

### Chaînes bornantes

Quand la chaîne dont on veut échantillonner la distribution stationnaire n'a pas la propriété de monotonie, il faut trouver des stratégies qui permettent de ne pas simuler tous les états.

**Algorithme 2:** Algorithme de Propp et Wilson pour le cas monotone [51]

**Données** : maj une fonction de mise à jour, $(N_k)_{k\in\mathbb{N}^*}$ une suite d'entiers positifs strictement croissante.

**début**
  $k \leftarrow 1$;
  **tant que** $X_{\min}(s) \neq X_{\max}(s)$ **faire**
    $X_{min}(s_{\min}) \leftarrow s_{\min}$ ;
    $X_{max}(s_{\max}) \leftarrow s_{\max}$ ;
    **pour** $n = -N_k + 1, \ldots, -N_{k-1}$ **faire**
      Choisir $u$ uniformément dans $[0, 1[$ ;
      $X_{\min}(s_{\min}) \leftarrow \text{maj}(X_{\min}(s_{\min}), u)$ ;
      $X_{\max}(s_{\max}) \leftarrow \text{maj}(X_{\max}(s_{\max}), u)$ ;
      $U[-n] \leftarrow u$;
    **pour** $n = -N_{k-1} + 1, \ldots, 0$ **faire**
      $u \leftarrow U[-n]$;
      $X_{\min}(s_{\min}) \leftarrow \text{maj}(X_{\min}(s_{\min}), u)$ ;
      $X_{\max}(s_{\max}) \leftarrow \text{maj}(X_{\max}(s_{\max}), u)$ ;
    $k \leftarrow k + 1$;
  **renvoyer** $X_{\min}(s_{\min})$.

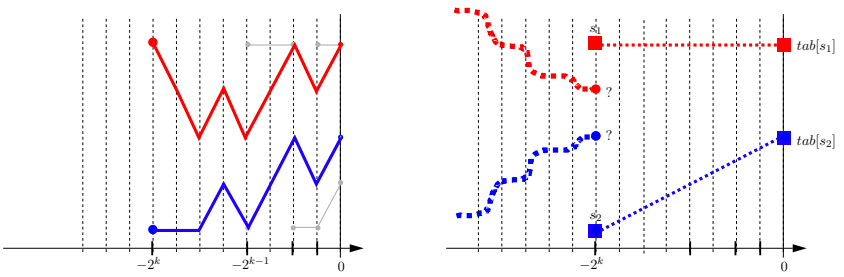



**FIGURE 1.6:** Propp et Wilson monotone.

Dans [37], Huber propose la technique des chaînes bornantes pour des espaces d'états dont les états sont des fonctions.

On suppose que l'espace des états de la chaîne $\mathcal{S}$ est contenu dans l'espace des applications de $V$ dans $C$ (*i.e.* $\mathcal{S} \subseteq C^V$). De manière générale, $|\mathcal{S}|$ croît exponentiellement en fonction de $|V|$, ce qui rend l'algorithme de simulation parfaite inutilisable en pratique. On donne ci-dessous un exemple d'un tel espace d'état que l'on gardera pour illustrer les chaînes bornantes.

**Coloration propre de graphe.** Soit $(V, E)$ un graphe non orienté de degré maximal $d$ dans lequel chaque nœud $v \in V$ peut prendre une couleur $c \in C := \{c_1, c_2, \ldots, c_k\}$. On appelle coloration du graphe, l'application qui attribue une couleur à chaque nœud du graphe. Notre intérêt se porte sur les **colorations propres** du graphe, c'est-à-dire les applications $\sigma : V \to C$ telles que

$$\forall (v, w) \in E,\ \sigma(v) \neq \sigma(w).$$

La figure 1.7 illustre une coloration propre pour un graphe contenant $5$ nœuds et $4$ couleurs, $C = \{\text{“}rouge\text{”}, \text{“}jaune\text{”}, \text{“}blanc\text{”}, \text{“}noir\text{”}\}$.

Dans [37], Huber présente deux définitions de chaîne bornantes qui sont équivalentes pour un espace d'états discret, comme c'est le cas ici. On ne présente ici que la première des deux

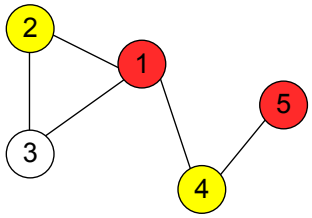



**FIGURE 1.7:** Exemple de coloration propre pour $|V| = 5$ et $|C| = 4$.

définitions (la forme 1). On considère deux chaînes de Markov : $(X_n)_n$ à valeurs dans $\mathcal{S}$, la chaîne que l'on veut borner et $(\mathcal{B}_n)_n$ à valeurs dans $\mathcal{P}(C)^{|V|}$ celle que l'on veut bornante.

Pour tout $n \in \mathbb{N}$, on a $X_n : V \to C$ et $\mathcal{B}_n : V \to \mathcal{P}(C)$. Si pour tout $v \in V$, $X_n(v) \in \mathcal{B}_n(v)$ alors on note $X_n \in_V \mathcal{B}_n$.

**Définition 2** (forme 1). *On dit que la chaîne de Markov $(\mathcal{B}_n)_n$ sur $\mathcal{P}(C)$ est une **chaîne bornante** de la chaîne de Markov $(X_n)_{n\in\mathbb{N}}$ sur $\mathcal{S}$, si*

$$X_n \in_V \mathcal{B}_n \implies X_{n+1} \in_V \mathcal{B}_{n+1}.$$

Remarquons que si $(\mathcal{B}_n)_n$ est une chaîne bornante de $(X_n)_n$ telle que $X_0 \in_V \mathcal{B}_0$ alors pour tout $n \geq 0$, $X_n \in_V \mathcal{B}_n$. Le nombre d'états bornés par $\mathcal{B}_n$ (*i.e.* le nombre d'applications $s$ vérifiant $s \in_V \mathcal{B}_n$) est au plus égal à $\prod_{v\in V} |\mathcal{B}_n(v)|$. Pour n'avoir plus qu'un seul état borné par $\mathcal{B}_n$, il faut donc vérifier que $|\mathcal{B}_n(v)| = 1$ pour tout $v \in V$. La vérification se fait avec un temps de calcul en $O(|V|)$. On construit la chaîne bornante $\mathcal{B}_n$ à partir d'une suite d'innovation $(U_n)_n$ et d'une fonction mise à jour $\mathrm{maj}$ de la chaîne $(X_n)_n$ de la façon suivante :

$$\begin{cases} X_0 = s \in \mathcal{S} \\ X_{n+1} = \mathrm{maj}(X_n, U_n) \quad \text{pour tout } n \geq 0, \end{cases}$$

et pour la chaîne bornante $(\mathcal{B}_n)_n$ :

- $\forall v \in V$ on pose $\mathcal{B}_0(v) = C$,
- $\forall v \in V,\ \mathcal{B}_{n+1}(v) = \bigcup_{s\in V^{\mathcal{B}_n(v)}} (\mathrm{maj}(s, U_n))$ pour tout $n \geq 0$.

On a ainsi $X_0 \in_V \mathcal{B}_0$ pour tout $s \in \mathcal{S}$. Par construction $(\mathcal{B}_n)_n$ est bien une suite bornante pour $(X_n)_n$. S'il existe un $n$ tel que pour tout $v \in V$, $|\mathcal{B}_n(v)| = 1$ alors pour $v \in V$ fixé, le seul élément $c$ de $\mathcal{B}_n(v)$ est aussi celui attribué par la configuration $X_n$ pour $v$. Dans ce cas, on a $X_n(v) = c$ et $\mathcal{B}_n(v) = \{c\}$. L'algorithme de simulation parfaite s'effectue en considérant la chaîne bornante à la place de toutes les trajectoires pour chaque $s \in \mathcal{S}$. Une telle chaîne $(B_n^N)_{n\in\mathbb{Z}}$ est définie comme suit :

- $\forall v \in V,\ B_N^N(v) = C$,
- $\forall v \in V,\ B_{n+1}^N(v) = \bigcup_{s\in V^{B_n^N(v)}} (\mathrm{maj}(s, U_n))$ pour tout $n \geq N$.

On obtient alors un algorithme de simulation parfaite en simulant la chaîne bornante et en considérant comme condition de fin de simulation : $\forall v \in V,\ |B_{n+1}^N(v)| = 1$. La chaîne bornante se représente avec une complexité mémoire en $O(|V| \times |C|)$. La complexité du calcul de sa fonction de mise à jour dépend de l'espace d'états considéré. On présente maintenant une chaîne bornante pour l'exemple de la coloration propre de graphe et l'algorithme de simulation parfaite utilisant cette chaîne.

**Simulation parfaite pour l'exemple de coloration de graphe.** La figure 1.8 illustre $X_0$ et $\mathcal{B}_0$. Dans la configuration $X_0 \in \mathcal{S}$, le nœud $1$ est colorié en jaune, nous avons donc $X_0(1) =$ "$jaune$". Pour chaque nœud $v \in V$, $\mathcal{B}_0(v)$ est un ensemble contenant les $4$ couleurs (jaune, rouge, noir et blanc). Ainsi on a $X_0 \in_V \mathcal{B}_0$.

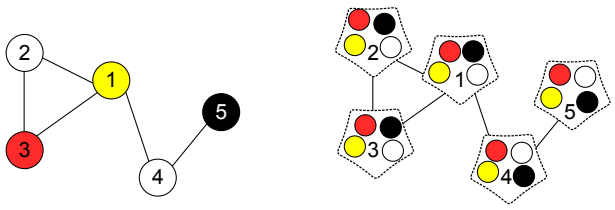



**FIGURE 1.8:** Configuration $X_0$ (à gauche) et chaîne bornante $\mathcal{B}_0$ (à droite).

Notons $|V|$ le nombre de nœuds et $b_s(v)$ le nombre de couleurs non utilisées par les voisins du nœud $v$ dans la configuration $s \in \mathcal{S}$. Soit $P$ la matrice de transition de chaîne $(X_n)_n$ définie telle que

$$P(s,s') = \begin{cases} \frac{1}{nb_s(v)} & \text{si } s(v) \neq g(v) \text{ et } \forall w \in V \setminus \{v\},\ s(w) = g(w) \\ \sum\limits_{v \in V} \frac{1}{nb_s(v)} & \text{si } s = g \\ 0 & \text{sinon.} \end{cases}$$

Cela signifie que $X_n$ et $X_{n+1}$ peuvent différer d'au plus une couleur. Comme nous l'avons vu dans la construction de la chaîne bornante, on a besoin de définir une fonction de mise à jour pour $(X_n)_n$. C'est ce que propose l'algorithme 3 qui est un échantillonneur de Gibbs [20]. On définit $\mathcal{N}_v$, l'ensemble des couleurs des nœuds voisins du nœud $v \in V$ pour la configuration $s \in \mathcal{S}$, autrement dit :

$$\mathcal{N}_v := \bigcup_{(w,v) \in E} s(w).$$

**Algorithme 3:** Échantillonneur de Gibbs.

**Données** : $s \in \mathcal{S}$
**début**
  Choisir uniformément $v \in V$;
  **faire**
    Choisir uniformément $c \in C$;
  **tant que** $c \in \mathcal{N}_v$;
  $s(v) \leftarrow c$;
  **renvoyer** $s$

Expliquons comment fonctionne cet échantillonneur de Gibbs. On considère $s \in \mathcal{S}$, une configuration du graphe. On commence par choisir un nœud uniformément parmi l'ensemble des nœuds du graphe. On choisit ensuite une couleur elle aussi uniformément parmi l'ensemble des couleurs. Si la couleur choisie appartient à l'ensemble $\mathcal{N}_v$ alors on procède à un nouveau choix uniforme de $c \in C$. On répète ceci jusqu'à obtenir une couleur qui n'appartient pas à l'ensemble des couleurs des voisins du nœud $v$. Quand on s'arrête, le nœud $v$ n'a donc plus aucun voisin ayant pour couleur $c$, on affecte alors au nœud $v$ la couleur $c$ et on retourne la nouvelle configuration.

L'algorithme 4 décrit la mise à jour de la chaîne bornante. On commence là encore par choisir un nœud $v$ uniformément parmi l'ensemble des nœuds de $V$. On efface toutes les couleurs potentielles du nœud $v$ en effectuant $\mathcal{B}(v) = \emptyset$. Puis on construit $\mathcal{B}(v)$ couleur par couleur de manière suivante : on choisit une couleur $c$ uniformément parmi l'ensemble des couleurs dans $C$. Si $c$ n'est dans aucun ensemble $\mathcal{B}(w)$ tel que $w$ soit voisin de $v$, (*i.e.* $c \notin \mathcal{N}_v$), alors $c$ est une couleur qui satisfait la contrainte de coloration propre pour toutes les configurations représentées par la chaîne bornante. On arrête alors de chercher d'autres couleurs pour $\mathcal{B}(v)$. Si $c \in \mathcal{N}_v$ et qu'il existe $w$ un voisin de $v$ tel que $\mathcal{B}(w) = \{c\}$ alors cette

**Algorithme 4:** Mise à jour de la chaîne bornante.

**Données** : $B$

**début**
  Choisir uniformément $v \in V$;
  $B(v) \leftarrow \emptyset$;
  **faire**
    Choisir uniformément $c \in C$;
    **si** *aucun voisin* $w$ *de* $v$ *est tel que* $B(w) = \{c\}$ **alors**
      $B(v) \leftarrow B(v) \cup \{c\}$;
  **tant que** $c \in \mathcal{N}_v$ ***ou*** $|B(v)| \leq d$;
  **renvoyer** $B$

couleur est déjà attribuée au nœud $w$, elle ne peut donc pas être attribuée au nœud $v$. Il faut donc continuer à chercher d'autres couleurs. Si $c \in \mathcal{N}_v$ et et que pour tout $w$ voisin de $v$ tel que $c \in \mathcal{B}(w)$ on a $|\mathcal{B}(w)| > 1$, $c$ est une couleur potentielle pour le nœud $v$, on l'ajoute donc à $\mathcal{B}(v)$. Quand $|\mathcal{B}(v)| > d$ on peut s'arrêter de choisir des couleurs. En effet, si $\mathcal{B}(v)$ contient $d+1$ couleurs, $v$ ayant au plus $d$ voisins, $\mathcal{B}(v)$ contient forcément au moins une couleur qui convient.

La mise à jour de la chaîne bornante se fait avec une complexité en $O(|C|d \log |C|)$. En effet, on tire au plus $|C|$ couleurs distinctes, pour chaque couleur tirée il faut regarder pour chaque voisin du nœud $v$ s'il possède la couleur tirée. Chaque voisin possède au plus $|C|$ couleurs, on vérifie avec une complexité en $\log(|C|)$ (pour $\mathcal{B}(w)$ encodé avec une structure de donnée adaptée) si le voisin $w$ possède la couleur tirée. Un nœud a au plus $d$ voisins.

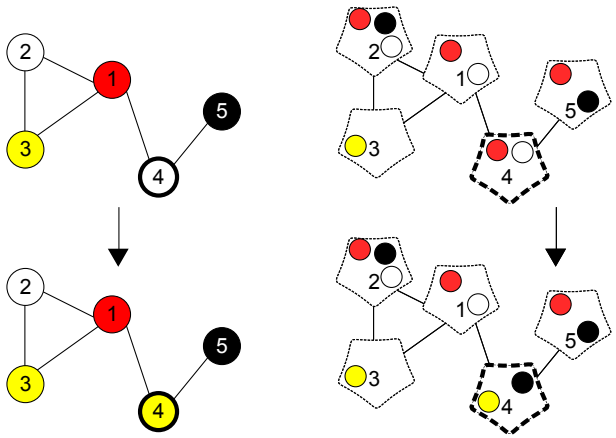



**FIGURE 1.9:** Mise à jour de la configuration $s$ (à gauche) et de la chaîne bornante $\mathcal{B}$ (à droite).

La figure 1.9 donne un exemple du fonctionnement des algorithmes 3 et 4 pour $v = 4$. Le dessin de droite illustre la mise à jour de la chaîne bornante. On commence par piocher la couleur "$noir$". Cette couleur appartient déjà à $\mathcal{B}(5)$ avec $|\mathcal{B}(5)| > 1$, on attribue donc "$noir$" à $\mathcal{B}(4)$. On pioche ensuite la couleur "$jaune$", comme cette dernière n'appartient ni à $\mathcal{B}(1)$ et ni à $\mathcal{B}(5)$, elle est attribuée à $\mathcal{B}(4)$ et l'algorithme s'arrête.

Toujours pour la figure 1.9, intéresserons-nous maintenant à la mise à jour de la configuration $s \in \mathcal{S}$ par l'échantillonneur de Gibbs et pour le même choix de nœud ($v = 4$) et de couleurs ("$noir$" puis "$jaune$"). La configuration $s \in \mathcal{S}$ est *inclue* dans la chaîne $\mathcal{B}$, en effet, pour chaque nœud $v \in V$ on a $s(v) \in \mathcal{B}(v)$. Pour la mise à jour de $s$, on commence par choisir le nœud $v = 4$ puis on regarde si ce nœud peut prendre la couleur "$noir$". C'est impossible car le nœud $5$ a déjà cette couleur. On regarde ensuite si le nœud $4$ peut prendre la couleur

"$jaune$". Ceci est tout a fait possible car aucun voisin du nœud $4$ ne porte la couleur "$jaune$". On attribue donc la couleur "$jaune$" au nœud $4$ et on a encore, après les mises à jour de la configuration $s$ et de la chaîne $\mathcal{B}$, $s(v) \in \mathcal{B}(v)$ pour tous les nœuds $v \in V$. On peut montrer que $\mathcal{B}$ est une chaîne bornante pour chaque configuration $s \in \mathcal{S}$.

Dans [37], Huber pousse encore plus loin l'utilisation des chaînes bornantes. Sous certaines hypothèses, il se sert des chaînes bornantes pour borner le temps de couplage. Par exemple pour la coloration propre de graphe, Huber donne une borne pour le temps de couplage si le nombre de couleurs $|C|$ satisfait $|C| \geq d(d+2)$.

**Théorème 3** (Huber [37]). *Si le nombre de couleurs $|C|$ est tel que $|C| \geq d(d+2)$, soit*

$$\beta = 1 - \frac{1-(d+1)d/(|C|-d+1)}{|V|}, \text{ alors } \mathbb{P}(\eta \leq \log_\beta(|V|) + \beta) \geq 1 - \beta^\theta.$$

## 1.2 Réseau fermé de files d'attente

On s'intéresse maintenant aux réseaux fermés de files d'attente [5, 21, 15]. Une **file** est constituée d'une file d'attente et d'un ou plusieurs serveurs. Un client qui arrive dans une file patiente dans la file d'attente jusqu'à être servi par un serveur puis quitte la file.

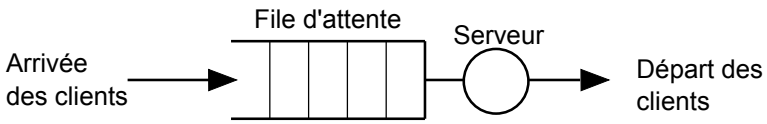



**FIGURE 1.10:** File simple à un serveur.

L'arrivée des clients est décrite par un processus stochastique. On appelle **capacité** d'une file le nombre maximal de clients que la file d'attente peut contenir. Si une file a atteint sa capacité maximale, tout nouveau client arrivant dans la file est rejeté. La capacité d'une file peut être finie ou infinie. Une **discipline de service** détermine l'ordre dans lequel les clients reçoivent le service. Par exemple, la discipline FIFO (First In First Out) implique que l'on serve les clients par ordre d'arrivée. La loi des arrivées ainsi que le temps de service sont modélisés par une loi de probabilité. La plus couramment utilisée est la loi exponentielle qui induit ainsi un processus de Poisson (on note ainsi M pour "markovien"). La notation de Kendall décrite ci-dessous, normalise la description d'une file.

T/X/S/C/Z
T : distribution des arrivées
X : distribution de service
S : nombre de serveurs
C : capacité de la file
Z : discipline de service

Par exemple une file $M/M/1$ a des temps d'inter-arrivée et de service exponentiels et indépendants ($M/M$), un seul serveur ($S = 1$). Quand des variables ne sont pas renseignées, cela signifie qu'elles prennent des valeurs par défaut. Pour $C$ non renseignée, cela signifie que la file a une capacité infinie ($C = \infty$) et $Z$ non renseigné signifie que la discipline de service est FIFO.

Un **réseau de files d'attente** est un ensemble de files interconnectées dans lequel les clients peuvent être dirigés après avoir été servis d'une file vers une autre. On dit qu'un réseau

est ouvert si les clients peuvent arriver de l'extérieur du réseau et en ressortir. Dans le cas opposé, si les clients ne sont autorisés ni à quitter le réseau ni à y rentrer on parle alors de **réseau fermé**. C'est ce type de réseaux que nous étudierons par la suite. Parmi les réseaux fermés, nous en distinguerons deux types : les réseaux monoclasses et multiclasses. Dans un réseau **multiclasse** les clients se partagent le réseau mais chaque client appartient à une classe et est servi puis dirigé en fonction de sa classe. Dans un réseau **monoclasse** on ne fait pas de distinction entre les clients.

### 1.2.1 Réseau monoclasse

#### Réseau de Gordon et Newell

On considère un réseau fermé muni de $K$ files d'attente $./M/1/\infty/1/FIFO$ contenant $M$ clients. Remarquons que l'on ne précise pas la distribution des arrivées car l'arrivée dans chaque file correspond à un départ dans une autre file.

On note $\mathcal{Q} := \{1, \dots, K\}$ l'ensemble des files du réseau. On appelle *réseau de Gordon et Newell* ou *réseau de Jackson fermé* [33, 38] un réseau dans lequel chaque file $k \in \mathcal{Q}$ possède une capacité de stockage infinie, un seul serveur avec taux de service égal à $\mu_k$ (*i.e.* service suivant une loi exponentielle de paramètre $\mu_k$) et une discipline FIFO. Les routages du réseau sont probabilistes. La probabilité pour un client venant d'être servi en file $i$ d'être dirigé en file $j$ est notée $p_{i,j}$. On note $\mathbf{P} = (p_{i,j})_{i,j\in\mathcal{Q}}$ la matrice des routages. Cette dernière est stochastique (pour tout $i \in \mathcal{Q}$, $\sum_{j\in\mathcal{Q}} p_{i,j} = 1$) et irréductible. On appelle **espace des états** toutes les configurations possibles du réseau, c'est-à-dire l'ensemble

$$\mathcal{S} = \left\{ \mathbf{x} = (x_1, x_2, \dots, x_K) \in \mathbb{N}^K \mid \sum_{k=1}^{K} x_k = M \right\}.$$

La valeur $x_k$ représente le nombre de clients en file $k$. Le nombre d'états du réseau est exactement $|\mathcal{S}| = \binom{K+M-1}{K-1} = \binom{K+M-1}{M}$. En effet, dénombrer les états est équivalent à compter le nombre de façons de repartir $M$ boules dans $K$ boîtes. Si on aligne les $M$ boules il faut $K-1$ bâtonnets pour séparer les $M$ boules en $K$ groupes. Cela revient donc à placer $M$ boules dans $M+K-1$ places ou de manière équivalente à placer $K-1$ bâtonnets dans $M+K-1$ places. Pour $K \ll M$, la cardinalité de $\mathcal{S}$ est en $O(M^K)$.

On modélise la dynamique d'un réseau de Gordon et Newell par une chaîne de Markov ergodique $(X_n)_{n\in\mathbb{N}}$ à valeur dans $\mathcal{S}$ telle que $\mathbb{P}(X_n = \mathbf{x} + (\mathbf{e_j} - \mathbf{e_i})\mathbb{1}_{\{x_i>0\}} | X_{n-1} = \mathbf{x}) = \mu_i p_{i,j}$.

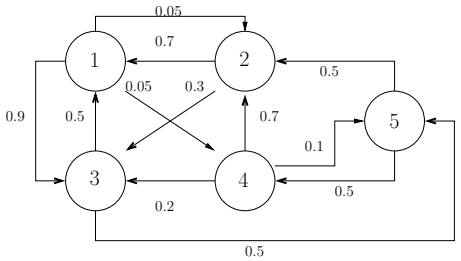



**FIGURE 1.11:** Réseau fermé avec 5 files.

On s'intéresse maintenant au régime stationnaire de ce type de réseau, c'est-à-dire à la distribution stationnaire de $(X_n)_{n\in\mathbb{N}}$. On appelle **taux de visite** de la file $k$ le nombre relatif

de passages normalisé dans une file $k$ entre deux passages par une file de référence. On le note $h_k$. Pour $k \in \mathcal{Q}$, les $h_k$ sont solutions des équations

$$h_k = \sum_{i=1}^{K} h_i p_{i,k} \quad \text{pour} \quad k \in \mathcal{Q} \text{ et } \sum_{k=1}^{K} h_k = 1.$$

Remarquons que le vecteur $\mathbf{h} = (h_1, \ldots, h_K)$ correspond à la distribution stationnaire d'une chaîne de Markov ayant comme matrice de transition la matrice $\mathbf{P}$.

**Théorème 4** (Forme produit d'un réseau de Gordon et Newell)**.** *La probabilité stationnaire d'un réseau de Gordon et Newell est donnée par*

$$\pi_{\mathbf{x}} = \frac{1}{G(K,M)} \prod_{k \in \mathcal{Q}} \left(\frac{h_k}{\mu_k}\right)^{x_k} \quad \textit{avec} \quad G(K,M) = \sum_{\mathbf{x} \in \mathcal{S}} \prod_{k \in \mathcal{Q}} \left(\frac{h_k}{\mu_k}\right)^{x_k}.$$

La valeur $G(K,M)$ est nommée **constante de normalisation**, grâce à elle on a $\sum_{k=1}^{K} \pi_{\mathbf{x}} = 1$. Dans [19], Buzen présente un algorithme qui calcule efficacement la constante de normalisation $G(K,M)$ en effectuant $O(KM)$ opérations. Notons

$$\mathcal{S}(k,m) = \left\{ \mathbf{x} = (x_1, x_2, \ldots, x_k) \in \mathbb{N}^k \mid \sum_{i=1}^{k} x_i = m \right\}.$$

L'idée est de constater que $\mathcal{S}(k,m)$ peut être divisé en deux sous-ensembles disjoints : l'ensemble des états dans lequel la dernière file ne contient aucun client ($x_k = 0$) et celui dans lequel la dernière file contient au moins un client. Notons respectivement $S_0(k,m)$ et $S_1(k,m)$ ces deux ensembles. On a alors $\mathcal{S}(k,m) = S_0(k,m) \cup S_1(k,m)$ et $S_0(k,m) \cap S_1(k,m) = \emptyset$. On en déduit ainsi :

$$\mathcal{S}(k,m) = \begin{cases} \{(0,\ldots,0)\} & \text{si } m = 0 \\ \{(m)\} & \text{si } k = 1 \\ \{\mathbf{x} + \mathbf{e_k} \mid \mathbf{x} \in \mathcal{S}(k,m-1)\} \cup \{(x_1,\ldots,x_{k-1},0) \mid \mathbf{x} \in \mathcal{S}(k-1,m)\} & \text{sinon.} \end{cases}$$

Cette relation de récurrence permet de calculer à l'aide d'un algorithme de programmation dynamique les constantes de normalisation $G(k,m)$ pour tout $m \in \{0,1,\ldots,M\}$ et $k \in \mathcal{Q}$. On pose alors :

- $G(k,0) = 1$ pour tout $k \in \mathcal{Q}$ ;
- $G(1,m) = \left(\frac{h_1}{\mu_1}\right)^m$ ;
- $G(k,m) = G(k,m-1)\frac{h_k}{\mu_k} + G(k-1,m)$.

Remarquons de plus que l'on peut exploiter la décomposition de $\mathcal{S}(k,m)$ pour générer l'espace des états $\mathcal{S} := \mathcal{S}(K,M)$.

Le théorème 4 s'étend également au réseau de Gordon et Newell pour lesquels les files peuvent posséder plusieurs serveurs. Supposons que chaque file $k \in \mathcal{Q}$ possède $E_k$ serveurs identiques, on a alors la propriété suivante.

**Propriété 1** (Extension Gordon et Newell)**.** *La probabilité stationnaire du réseau est telle que*

$$\pi_{\mathbf{x}} = \frac{1}{G(K,M)} \prod_{i \in \mathcal{Q}} f_i(x_i) \textit{ avec } f_i(x_i) = \frac{1}{\prod_{m=1}^{x_i} \min(m, E_i)} \left(\frac{h_i}{\mu_i}\right)^{x_i} \textit{ et } G(K,M) = \sum_{\mathbf{x} \in \mathcal{S}} \prod_{k \in \mathcal{Q}} f_i(x_i).$$

Dans [19], Buzen calcule la constante de normalisation pour dans le cas multi-serveurs en effectuant $2KM(M+1)$ opérations. Le résultat de Gordon et Newell s'étend plus générale-

ment au cas des files ayant un taux de service dépendant de l'état de la file. Soit $\mu_i(m)$ le taux de service de la file $i$ possédant $m$ clients, on pose alors

$$f_i(x_i) = \frac{{h_i}^{x_i}}{\prod_{m=1}^{x_i} \mu_i(m)}.$$

#### Capacité finie

Quand une file est à capacité finie et qu'elle atteint sa capacité maximale, le service est interrompu. Ceci crée un blocage. Dans [47] Onvural fait l'inventaire des différents type de blocages que l'on peut rencontrer. Celui que nous considérerons dans le chapitre 3 est connu sous le nom de RS-RD (*repetitive service - random destination*). Quand un client qui vient d'être servi est dirigé vers une file qui est pleine, il retourne dans la file d'attente qui vient de le servir et perd son service. Pour son prochain service, il sera routé indépendamment la première destination. Les réseaux fermés possédant de telles files bénéficient d'une distribution stationnaire à forme produit si le routage est réversible [2], c'est-à-dire si pour tout couple de file $(i, j)$ la probabilité du routage de $i$ vers $j$ est identique à celle de $j$ vers $i$. Dans le cas contraire, le calcul de la distribution stationnaire est impossible au vu de la cardinalité de l'espace des états.

Pour échantillonner la distribution stationnaire on peut par exemple utiliser la méthode de Monte-Carlo (en sachant qu'on obtiendra des échantillons biaisés). Le grand nombre d'états ne permet pas en outre l'application naïve de l'algorithme de simulation parfaite. De plus, on verra dans la section 1.4 que la contrainte globale sur le nombre de clients rend inutilisable les stratégies pour la simulation parfaite présentées en section 1.1.5. On présentera au chapitre 3 une nouvelle technique permettant la simulation parfaite des réseaux fermés de files d'attente monoclasse à capacité finie.

### 1.2.2 Réseau multiclasse

On considère un réseau fermé multiclasse possédant $Z$ classes de clients et $M_z$ clients de chaque classe $z \in \{1, \dots, Z\}$. On note $\mathbf{M} = (M_1, \dots, M_Z)$ le vecteur du nombre de clients pour chaque classe. Un état du réseau est représenté par une matrice $\mathbf{x} = (x_{z,k}) \in \mathbb{N}^{Z \times K}$ dans laquelle les coefficients $x_{z,k}$ ont pour valeur le nombre de clients de classe $z$ en file $k$. Ainsi pour chaque classe $z$ fixée, on a $\sum_{k=1}^{K} x_{z,k} = M_z$. On en déduit

$$\mathcal{S} = \left\{ \mathbf{x} \in \mathbb{N}^{ZK} \mid \forall z, \sum_{k=1}^{K} x_{z,k} = M_z \right\} \quad \text{et} \quad |\mathcal{S}| = \prod_{z=1}^{Z} \binom{K-1}{K+M_z-1}.$$

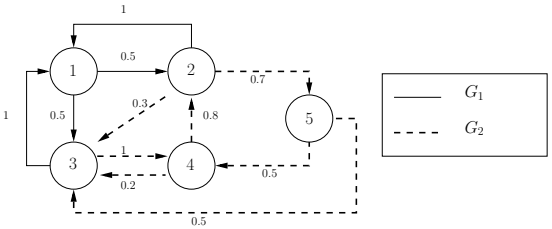



**FIGURE 1.12:** Réseau multiclasse $Z = 2$.

On appelle **réseaux BCMP** les réseaux référencés dans le tableau ci-dessous. Ils furent introduits par Baskett, Chandy, Muntz et Palacios dans [4]. Comme pour le cas des réseaux

monoclasses, ces réseaux possèdent une constante de normalisation $G(K, \mathbf{M})_T$ (dépendant du type de réseau $T$). Elle se calcule à l'aide d'une relation de récurrence. Il y a quatre types de réseau BCMP,

| Type | Discipline de service | Lois de service |
|---|---|---|
| 1 | FIFO (premier arrivé, premier servi) | Exponentielle, indépendante de la classe en service : $\mu_i$ taux de service du serveur $i$ |
| 2 | PS (temps partagé) | Générales différentes pour chaque classe (à la transformée de Laplace rationnelle) : $\frac{1}{\mu_{ir}}$ temps moyen des clients de classe $r$ (serveur $i$) |
| 3 | IS (nombre de serveurs infini) | |
| 4 | LCFS-PR (dernier arrivé, premier servi, avec préemption du service) | |

Une grande partie des réseaux multiclasses non-référencés dans le tableau ci-dessus ne possèdent pas la forme produit. Le chapitre 4 s'intéresse à la simulation parfaite d'une famille de réseaux ne possédant pas la forme produit.

### 1.2.3 Synchronisation

Dans le chapitre 5, on s'intéresse à des réseaux monoclasses qui possèdent des serveurs synchronisant des clients de files différentes et d'autre dédoublant les clients une fois servis pour les envoyer dans des files différentes. Ceci correspond à des modèles de type *fork–join* qui sont notamment utilisés pour modéliser l'exécution en parallèle de programmes informatique. La figure ci-dessous en illustre un exemple. Les serveurs 1, 2, 3 et 4 exécutent des tâches en parallèle, le serveur 5 effectue la fusion *join* et le serveur 7 la redistribution des tâches (*fork*). Par exemple si aucune des files 3 et 6 n'est vide lors du service du serveur 5, ce dernier sert les files 3 et 6 simultanément. Les deux clients servis sont fusionnés en un seul et ce dernier est dirigé en file 7.

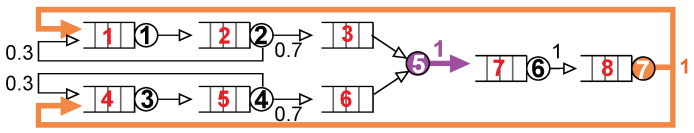



**FIGURE 1.13:** Réseau avec synchronisations.

Un réseau de files synchronisées peut être vu comme un réseau de Petri [49]. Les files sont alors appelées **places**, les serveurs des **transitions**, les clients des **jetons** et le vecteur $\mathbf{x} = (x_1, \ldots, x_K)$ le **marquage**. Les transitions du réseau sont numérotées de 1 à $T$ et les places de 1 à $K$. Dans un réseau de Pétri, on note $c_{k,i}$ le nombre de jetons ajoutés (ou retranchés) à la place $k$ par le déclenchement de la transition $i$. On décrit la dynamique du réseau par la matrice $C = (c_{k,i})$ pour $k = 1, \ldots K$ et $i = 1 \ldots K$. Le réseau est dit **temporisé** si on décrit pour chaque transition la loi du temps entre deux déclenchements successifs.

## 1.3 Simulation parfaite et files d'attente

On s'intéresse maintenant aux travaux alliant simulation parfaite et réseaux de files d'attente. Les travaux [56, 57] abordent la simulation parfaite de réseaux ouverts dont les files ont une capacité finie, l'auteur exploite la monotonie des transitions pour parvenir à effectuer la simulation parfaite. Dans [17, 16] ces résultats sont étendus aux réseaux ouverts pouvant posséder des capacités de files infinies. L'idée est alors de borner certains événements. On commence dans cette section, par illustrer la technique monotone ainsi que celle des enveloppes sur quelques exemples de réseaux très simples. On expliquera dans la section suivante pourquoi ces techniques ne sont pas applicables aux cas des réseaux fermés de files d'attente.

### 1.3.1 Simulation d'un réseau monotone

On considère pour modèle un réseau ouvert composé de 4 files d'attente $./M/1/C$. Le réseau est illustré ci-dessous (figure 1.14). Chaque file d'attente possède un taux de service $\mu_k$ et une capacité finie $C$. On considère un blocage *RS-RD*, c'est-à-dire que si la file d'arrivée est pleine, le client retourne dans sa file d'origine et perd son service. Les clients ne peuvent entrer dans le réseau que par la file 1 et en ressortir qu'après avoir été servis en file 4. À chaque instant $n$ il peut donc se produire trois types d'événements : un client arrive de l'extérieur, un client quitte le réseau et un client qui vient d'être servi est dirigé vers une autre file.

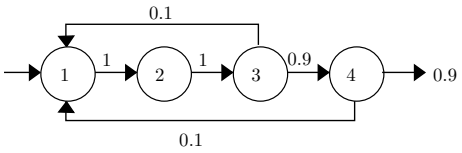



**FIGURE 1.14:** Réseau ouvert avec 4 files.

Les arrivées en file 1 suivent un processus de Poisson d'intensité $\lambda$. On suppose que $\lambda + \mu_1 + \mu_2 + \mu_3 + \mu_4 = 1$. La dynamique de ce système est modélisée par une chaîne de Markov $(X_n)_{n\in\mathbb{N}}$ à valeur dans $\mathcal{S}$ avec

$$\mathcal{S} = \{(x_1, x_2, x_3, x_4) \mid \forall k,\ 0 \leq x_k \leq C\}.$$

On en déduit ainsi $|\mathcal{S}| = (C+1)^4$. Soit $\mathrm{maj} : \mathcal{S} \times [0,1[ \to \mathcal{S}$ une fonction de mise à jour valide de la chaîne donnée par le tableau 1.2.

| $\mathrm{maj}(\mathbf{x}, u)$ | Condition sur $u$ | Modélise |
|---|---|---|
| $\mathbf{x} + \mathbb{1}_{\{x_1<C\}}\mathbf{e_1}$ | $u < \lambda$ | Arrivée d'un nouveau client |
| $\mathbf{x} + \mathbb{1}_{\{x_1>0, x_2<C\}}(\mathbf{e_2} - \mathbf{e_1})$ | $\lambda \leq u < \lambda + \mu_1$ | Service file 1 |
| $\mathbf{x} + \mathbb{1}_{\{x_2>0, x_3<C\}}(\mathbf{e_3} - \mathbf{e_2})$ | $\lambda + \mu_1 \leq u < \lambda + +\mu_1 + \mu_2$ | Service file 2 |
| $\mathbf{x} + \mathbb{1}_{\{x_3>0, x_4<C\}}(\mathbf{e_4} - \mathbf{e_3})$ | $\lambda + \mu_1 + \mu_2 \leq u < \lambda + \mu_1 + \mu_2 + \frac{9\mu_3}{10}$ | Service file 3, routage file 4 |
| $\mathbf{x} + \mathbb{1}_{\{x_3>0, x_1<C\}}(\mathbf{e_1} - \mathbf{e_3})$ | $\lambda + \mu_1 + \mu_2 + \frac{9\mu_3}{10} \leq u < \lambda + \mu_1 + \mu_2 + \mu_3$ | service file 3, routage file 1 |
| $\mathbf{x} + \mathbb{1}_{\{x_4>0, x_1<C\}}(\mathbf{e_1} - \mathbf{e_4})$ | $\lambda + \mu_1 + \mu_2 + \mu_3 \leq u < \lambda + \mu_1 + \mu_2 + \mu_3 + \frac{\mu_4}{10}$ | Service file 4, routage file 1 |
| $\mathbf{x} - \mathbb{1}_{\{x_4>0\}}\mathbf{e_4}$ | $\lambda + \mu_1 + \mu_2 + \mu_3 + \frac{\mu_4}{10} \leq u < 1$ | service file 4, routage extérieur |

**TABLE 1.2:** Description de la fonction de mise à jour du réseau monotone.

Pour pouvoir utiliser la technique monotone il faut commencer par exhiber un ordre sur l'espace des états. Considérons l'*ordre composante par composante* définit tel que $\mathbf{x} \preceq_c \mathbf{y}$ si et seulement si pour tout $k \in \{1,2,3,4\}$ on a $x_k \leq y_k$.

**Lemme 1.** *La chaîne de Markov $(X_n)_{n\in\mathbb{N}}$ est monotone selon l'ordre $\preceq_c$.*

Chaque valeur $u$ correspond à un événement se produisant dans le réseau. Pour montrer le lemme, il suffit donc de montrer que pour tout $u \in [0,1[$, la fonction de mise à jour $\mathrm{maj}$ est monotone, c'est-à-dire que pour tout $\mathbf{x},\ \mathbf{y} \in \mathcal{S}$ si $\mathbf{x} \preceq_c \mathbf{y}$ alors $\mathrm{maj}(\mathbf{x}, u) \preceq_c \mathrm{maj}(\mathbf{y}, u)$.

Notons $\mathbf{x}_{\min} := \min(\{\mathbf{x} \mid \mathbf{x} \in \mathcal{S}\}) = (0,0,0,0)$ et $\mathbf{x}_{\max} := \max(\{\mathbf{x} \mid \mathbf{x} \in \mathcal{S}\}) = (C,C,C,C)$ alors on peut appliquer la simulation parfaite en considérant uniquement les trajectoires issues de $\mathbf{x}_{\min}$ et de $\mathbf{x}_{\max}$.

### 1.3.2 Technique des enveloppes

On s'intéresse à un modèle de réseau ouvert de files d'attente non monotone et on applique la technique des enveloppes présentée dans [18].

**Arrivées *par lots*** Considérons une file d'attente $M/M/1/C$ ayant la particularité de recevoir un lot de clients dont la taille peut varier de 1 à $C$ (la capacité de la file) et dont l'intégralité du lot est rejeté si il n'y a pas la place pour lui. L'espace des états est donné par $\mathcal{S} = \{0, 1, \ldots, C\}$. Ce genre modèle n'est pas monotone. En effet, soient $x_1 = C - 1$ et $x_2 = C - 2$ deux états de la file. On a $x_1 > x_2$. Supposons un événement $u \in [0, 1[$ tel que $u$ corresponde à l'arrivée d'un lot de 2 clients arrive dans la file, alors $\mathrm{maj}(x_1, u) = x_1$ et $\mathrm{maj}(x_2, u) = C$ et on a donc $\mathrm{maj}(x_1, u) < \mathrm{maj}(x_2, u)$. On ne peut donc pas appliquer la technique monotone présentée en sous-section 1.1.5. Soient $b \in \mathcal{S}$ et $t \in \mathcal{S}$ tels que $b \leq t$, posons maintenant pour tout $u \in [0, 1[$

$$b' = \underline{\mathrm{maj}}(b, t, u) := \inf_{b \leq x \leq t} \mathrm{maj}(x, u) \qquad \text{et} \qquad t' = \overline{\mathrm{maj}}(b, t, u) := \sup_{b \leq x \leq t} \mathrm{maj}(x, u). \tag{1.3}$$

On a alors pour tout $u \in [0, 1[$, $b' \leq t'$ et le couple $(b', t')$ forme la nouvelle enveloppe. Le calcul de $b'$ et $t'$ dans 1.3 se fait sans calculer toutes les $\mathrm{maj}(x, u)$ pour $b \leq x \leq t$.

Posons $b_0 := \min(\mathcal{S}) = 0$, $t_0 := \max(\mathcal{S}) = C$ et pour tout $n \geq 1$,

$$b_n = \underline{\mathrm{maj}}(b_{n-1}, t_{n-1}, u) \qquad \text{et} \qquad t_n = \overline{\mathrm{maj}}(b_{n-1}, t_{n-1}, u).$$

Le temps de couplage est alors donné par $\eta := \inf\{n \in \mathbb{N} \mid b_n = t_n\}$.

On appelle enveloppe à l'instant $n$, l'ensemble des états compris entre $b_n$ et $t_n$. De façon générale, si le calcul de la mise à jour de l'enveloppe peut se faire efficacement, c'est-à-dire sans avoir besoin d'examiner tous les états compris entre $b$ et $t$ (et dans un temps raisonnable), alors on peut se servir de l'enveloppe comme substitut à l'ensemble de tous les états. L'algorithme de simulation parfaite ayant recours une enveloppe a pour condition d'arrêt : $b_n = t_n$. Sa complexité en temps est en $O(\eta C_{env})$ avec $\eta$ le temps de couplage et $C_{env}$ la complexité du calcul de $b_n$ et $t_n$.

Dans [18] les auteurs appliquent la technique des enveloppes dans un réseau ouvert de files d'attente à $K$ files. Notamment ils montrent que le calcul des $b_n$ et $t_n$ s'effectue avec une complexité en temps de calcul en $O(K \log C)$.

### 1.3.3 Schéma d'approximation en temps polynomial

Dans [41], Kijima et Matsui proposent un schéma d'approximation randomisé entièrement en temps polynomial (FPRAS) afin de calculer la constante de normalisation $G(K, M)$ d'un réseau de Gordon et Newell d'un réseau fermé de files d'attente à $K$ files et $M$ clients à plusieurs serveurs. L'algorithme requiert des échantillons $\mathbf{x}$ dont la loi correspond à la distribution stationnaire d'un réseau fermés de files d'attente à $k$ files et $m$ clients tel que $m \leq M$ et $k \leq K$ et ayant une file plus chargée que les autres. Ils se servent de la forme produit pour modifier la structure de la chaîne : une file $i < k$ est tirée au hasard et les clients des files $i$ et $i+1$ sont reparties dans chacune de ses files en fonction de la distribution stationnaire aux coordonnées $i$ et $i + 1$. Cela peut être obtenu avec un temps de couplage en $O(K^3 \log(M))$. Comme les files $i$ et $i + 1$ sont adjacentes, la chaîne devient monotone en utilisant l'ordre des sommes cumulées. On peut alors faire simulation parfaite. Cette chaîne est intrinsèquement liée à la forme produit et ne peut donc pas être utilisée pour simuler des réseaux fermés plus généraux.

### 1.3.4 Simulation parfaite des réseaux de Petri

Dans [14], les auteurs montrent comment construire un algorithme de simulation parfaite pour des réseaux de Petri à choix libre [23] dans le cas de réseaux markoviens et non markoviens. Pour les réseaux markoviens, les auteurs ont recours à un ensemble de marquages dits extrémaux et jouant le rôle d'enveloppe.

Dans [3], Balsamo et ali. présentent un algorithme de simulation parfaite pour des réseaux de Petri temporisés. Ils représentent l'ensemble de tous les marquages par un arbre de décision multivarié (MDD) et réalisent la simulation directement sur le MDD.

## 1.4 Problématique pour les réseaux fermés de files d'attente

Dans cette section, on illustre pour exemple simple de réseau fermé la difficulté d'utiliser l'algorithme de simulation parfaite quand le réseau considéré n'a pas de forme produit.

### 1.4.1 Modèle

Reprenons pour exemple le réseau de la figure 1.14 pour lequel on supprime l'arrivée en provenance de l'extérieur du réseau (arrivée file 1) et la sortie du réseau (sortie file 4), ceci donne le réseau de figure 1.15.

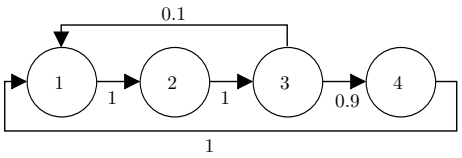



**FIGURE 1.15:** Réseau fermé avec 4 files.

Considérons que ce réseau contient 3 clients au total et notons $t_{i,j} : \mathcal{S} \to \mathcal{S}$ la fonction qui réalise un routage de la file $i$ vers la file $j$, on a alors :

$$t_{i,j}(\mathbf{x}) = \mathbf{x} + (\mathbf{e_j} - \mathbf{e_i})\mathbb{1}_{\{x_i>0 \text{ et } x_j<C_j\}}.$$

Soit $u$ tel que $\mathrm{maj}(\mathbf{x}, u)$ corresponde au service en file $i$ et à l'ordre de routage vers la file $j$, ainsi on a $\mathrm{maj}(\mathbf{x}, u) = t_{i,j}(\mathbf{x})$.

### 1.4.2 Technique monotone

La contrainte sur le nombre total de clients rend impossible l'obtention d'un maximum et d'un minimum pour l'ordre *composante par composante*. Considérons maintenant un ordre total, l'*ordre lexicographique*, noté $\prec_l$ et défini tel que $\mathbf{x} \prec_l \mathbf{y}$ si et seulement s'il existe $k' \leq 4$ tel que pour tout $k < k'$ on ait $x_k = y_k$ et $x_{k'} < y_{k'}$. Les états extrémaux sont donc $\mathbf{x}_{\min} = (0, 0, 0, 3)$ et $\mathbf{x}_{\max} = (3, 0, 0, 0)$. En appliquant trois fois la transition de la file 1 vers la file 2 puis trois fois encore la transition de la file 4 vers la file 1, on a :

$$t^3_{4,1} \circ t^3_{1,2}(\mathbf{x}_{\min}) = (3, 0, 0, 0) = \mathbf{x}'_{\min} >_l t^3_{4,1} \circ t^3_{1,2}(\mathbf{x}_{\max}) = (0, 3, 0, 0) = \mathbf{x}'_{\max}$$

Ainsi il existe une suite de transitions ayant une probabilité non nulle d'apparition transformant $\mathbf{x}_{\min}$ en $\mathbf{x}'_{\min}$ et $\mathbf{x}_{\max}$ en $\mathbf{x}'_{\max}$ tels que $\mathbf{x}'_{\max} \prec_l \mathbf{x}'_{\min}$. La chaîne associée à ce modèle n'est donc pas monotone pour l'ordre lexicographique.

Définissons maintenant un autre ordre, celui des *sommes cumulées*, il est partiel. On note $\mathbf{x} \preceq_s \mathbf{y}$ si et seulement si pour tout $k \in \{1,2,3,4\}$ on a $\sum_{i=1}^{k} x_i \leq \sum_{i=1}^{k} y_i$. Les états extrémaux sont encore $\mathbf{x}_{\min} = (0,0,0,3)$ et $\mathbf{x}_{\max} = (3,0,0,0)$. Et là encore, la chaîne n'est pas monotone pour l'ordre des *sommes cumulées*. En effet en prenant la même suite de transition que précédemment, on obtient $\mathbf{x}'_{\max} \prec_s \mathbf{x}'_{\min}$.

Nous venons de voir qu'il n'est pas facile de trouver un ordre sur les états ou les majorants, si tant est qu'ils existent. Regardons maintenant pourquoi la technique des chaînes bornantes ne peut pas s'appliquer.

### 1.4.3 Technique des chaînes bornantes

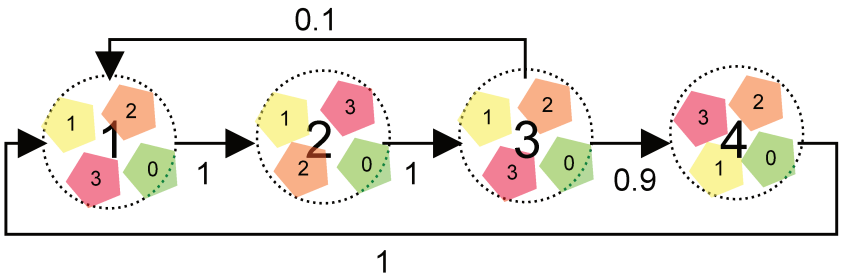



**FIGURE 1.16:** Initialisation de la chaîne bornante.

Essayons maintenant la technique des chaînes bornantes de Huber [37]. Un état $\mathbf{x} \in \mathcal{S}$ peut être vu comme une configuration $\mathbf{x} : \{1,2,\ldots,K\} \to \{0,1,\ldots,M\}$ qui à une file $k$ attribue un nombre de clients $x_i$. Pour faire le parallèle avec la coloration propre du graphe, le graphe est alors le réseau, les files sont les nœuds et le nombre de clients est la couleur du nœud. L'initialisation de la chaîne $\mathcal{B}$ qui représente tous les états $\mathbf{x} \in \mathcal{S}$ est illustrée en figure 1.16. Pour tout $k \in \{1,\ldots,K\}$ on pose $\mathcal{B}_0(k) = \{0,1,\ldots,M\}$. On peut encoder cette chaîne avec une complexité en mémoire en $O(KM)$. Le problème ne réside donc pas dans l'encodage de la chaîne mais dans la définition de la mise à jour de cette chaîne. Comme nous l'avons vu, pour définir une mise à jour il faut s'assurer qu'après une transition $t_{i,j}$, tous les états appartenant à l'ensemble $\{t_{i,j}(\mathbf{x}) \mid \mathbf{x} \in \mathcal{S}\}$ soient représentés par la mise à jour de la chaîne. Cependant dans l'ensemble $\{t_{i,j}(\mathbf{x}) \mid \mathbf{x} \in \mathcal{S}\}$ il existe des états tels que $x_i = \{0,1,\ldots,M\}$ et $x_j = \{0,1,\ldots,M\}$ ainsi quelque soit la transition $t_{i,j}$, pour tout $k \in \{1,\ldots,K\}$ on a $\mathcal{B}_1(k) = \{0,1,\ldots,M\}$.

Par exemple, imaginons que nous voulions appliquer la transition $t_{3,1}$ sur la chaîne bornante de la figure 1.16. Alors on ne pourrait pas enlever l'étiquette 3 dans la file 3 car la file 1 possède une étiquette 0 et l'état $(0,0,3,0)$ appartient à l'ensemble des états représentés par la chaîne. De plus, à cause des états $(0,3,0,0)$ et $(0,0,0,3)$ et parce que la file 3 possède l'étiquette 0, on ne pourrait pas retirer l'étiquette 0 de la file 1. De façon générale, la chaîne bornante n'est modifiable par aucune des mises à jour du système.

### 1.4.4 Enveloppes

Essayons la technique des enveloppes en considérant l'ordre l'ordre lexicographique $\preceq_l$ qui est un ordre total.

Pour effectuer une mise à jour qui correspond au routage de la file $i$ vers la file $j$ on calcule

$$\mathbf{b}_{n+1} = \inf_{\mathbf{b}_n \preceq_l \mathbf{x} \preceq_l \mathbf{t}_n} t_{i,j}(\mathbf{x}) \qquad \text{et} \qquad \mathbf{t}_{n+1} = \sup_{\mathbf{b}_n \preceq_l \mathbf{x} \preceq_l \mathbf{t}_n} t_{i,j}(\mathbf{x}).$$

Pour commencer la simulation, on a $\mathbf{b}_0 = (0,0,0,3) \in \mathcal{S}$ et $\mathbf{t}_0 = (3,0,0,0) \in \mathcal{S}$, l'enveloppe représente donc les $20$ états de l'espace d'états $\mathcal{S}$. Seules les transitions $t_{1,2}$ et $t_{4,1}$ affectent les états $\mathbf{b}_0$ et $\mathbf{t}_0$. Par exemple on a

$$\mathbf{b}_1 = \inf_{\mathbf{b}_0 \preceq_l \mathbf{x} \preceq_l \mathbf{t}_0} t_{1,2}(\mathbf{x}) = (0,0,0,3) \qquad \text{et} \qquad \mathbf{t}_1 = \sup_{\mathbf{b}_0 \preceq_l \mathbf{x} \preceq_l \mathbf{t}_0} t_{1,2}(\mathbf{x}) = (2,1,0,0).$$

De plus, après la transition $t_{1,2}$, l'enveloppe représente donc $19$ états tandis qu'on a $|t_{1,2}(\mathcal{S})| = 14$. De plus, certaines transitions peuvent faire grandir le nombre d'états représentés par l'enveloppe de façon non négligeable. Par exemple l'enveloppe telle que $\mathbf{b}_n = (0,0,2,1)$ et $\mathbf{t}_n = (0,3,0,0)$ représente $8$ états tandis que l'enveloppe $\mathbf{b}_{n+1} = (0,0,3,0)$ et $\mathbf{t}_{n+1} = (1,2,0,0)$ obtenue après la transition $t_{4,1}$ représente $13$ états, ceci représente une augmentation de $5$ états soit $25\%$ de la taille de l'espace des états.

Le tableau ci-dessous donne les temps de couplage par l'avant : exact et en utilisant la technique des enveloppes pour 10 simulations. Pour chaque couple de simulation (Exact et Enveloppe) on a utilisé la même suite d'innovation. On constate que le temps de couplage en utilisant les enveloppes est titanesque en comparaison du temps de couplage exact.

| Temps de couplage | | |
|---|---|---|
| Simulation | Exact | Enveloppe |
| 1 | 22 | 36946 |
| 2 | 46 | 9878 |
| 3 | 47 | 3315 |
| 4 | 17 | 1367 |
| 5 | 35 | 8466 |
| 6 | 17 | 27652 |
| 7 | 20 | 13109 |
| 8 | 21 | 26990 |
| 9 | 12 | 10328 |
| 10 | 23 | 5421 |

**FIGURE 1.17:** Comparaison des temps de couplage par l'avant.

Considérons maintenant un autre ordre, l'ordre des sommes cumulées $\preceq_s$. Cet ordre est partiel car tous les éléments ne sont pas comparables. On obtient comme pour l'ordre lexicographique, $\mathbf{b}_0 = (0,0,0,3)$ et $\mathbf{t}_0 = (3,0,0,0)$. Comme dans le cas précédent, seules les transitions $t_{1,2}$ et $t_{4,1}$ affectent l'enveloppe représentée par le couple $(\mathbf{b}_0, \mathbf{t}_0)$.

Considérons comme premier pas de simulation, la transition $t_{4,1}$. Ainsi on obtient comme borne inférieure $\mathbf{b}_1 = (0,0,1,2)$ et comme borne supérieure $\mathbf{t}_1 = (3,0,0,0)$. Remarquons que l'état $(0,0,1,2)$ n'appartient pas à $E_1 := \{t_{4,1}(\mathbf{x}) \mid \mathbf{x} \in \mathcal{S}\}$, l'enveloppe $(\mathbf{b}_1, \mathbf{t}_1)$ représente donc plus d'états que ceux contenus dans $E_1$, elle en représente 19. Ceci laisse aussi présager, comme pour l'ordre lexicographique, un temps de couplage trop grand. Mais surtout, ce qui pose problème c'est le calcul de la fonction de mise à jour. Par exemple, les états $(1,0,0,2)$, $(0,0,3,0)$ et $(1,0,0,2)$ minorent $E_1$ mais sont incomparables pour l'ordre des sommes cumulées, c'est pour cela qu'on a obtenu $b_1 = (0,0,1,2)$. Considérer tout les états représentés par l'enveloppe, les comparer et en déduire une borne inférieure n'est pas une solution efficace pour le calcul de la mise à jour de l'enveloppe. Pour pouvoir envisager d'utiliser la technique des enveloppes en utilisant l'ordre ses sommes cumulée, il faudrait trouver un algorithme qui calcule l'enveloppe de manière efficace. Ceci semble être un problème combinatoire complexe qui de plus ne garantira pas de trouver un temps de couplage relativement proche du temps de couplage exact. Dans le chapitre 3 on proposera une autre structure de données jouant le rôle d'enveloppe et permettant la mise à jour en temps raisonnable.

### 1.4.5 Réseaux en anneau et états extrémaux

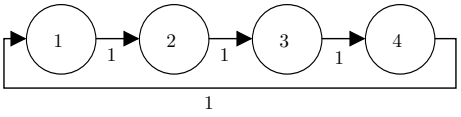



**FIGURE 1.18:** Réseau en anneau avec 4 files.

Si on considère le réseau en anneau défini tel que $p_{i,j} = 1$ avec $j = (i+1) \mod K$ (voir figure 1.18), alors dans ce type de réseau particulier, Bouillard et Gaujal ont montré dans [14] qu'il existe un ensemble d'états $S_{ext} \subseteq \mathcal{S}$ de cardinal $K$ appelés **états extrémaux** tel que pour toute suite de transitions $t$ on ait $|t(\mathcal{S})| = 1$ si et seulement si $|t(S_{ext})| = 1$. L'ensemble des états extrémaux correspond à l'ensemble des états pour lesquels les clients sont concentrés dans une seule file. Dans le modèle de la figure 1.18 si le réseau contient 3 clients alors l'ensemble des états extrémaux est donné par $S_{ext} = \{(0,0,0,3),(0,0,3,0),(0,3,0,0),(3,0,0,0)\}$. Ce résultat permet dans un réseau en anneau de trouver le temps de couplage exact de tel réseaux en simulant seulement $K$ trajectoires au lieu de $|\mathcal{S}|$. Il nous permettra d'effectuer des comparaisons de temps de couplage pour la technique développée dans cette thèse pour les réseaux fermé de files d'attente. Notons cependant que l'utilisation des états extrémaux est adapté au cas des graphes d'événements et donc en ce qui nous concerne ne convient qu'aux réseaux en anneau.

### 1.4.6 Conclusion

On a rappelé quelques techniques de simulations parfaites de chaîne de Markov, en particulier la technique des enveloppes a été utilisée sur un réseau ouvert de files d'attentes. La simulation parfaite de réseaux fermés de files d'attente est un problème difficile essentiellement à cause de la contrainte globale sur le nombre de clients. En effet, cette dernière laisse présager qu'il n'existe pas d'ordre partiel (ou total) facile à mettre en œuvre qui rendrait la chaîne monotone. De plus, elle rend inutilisable la technique des chaînes bornantes [37]. La technique des enveloppes fonctionne en théorie, mais laisse présager un temps de couplage bien trop grand.

Dans le chapitre 3 on présentera une nouvelle technique de simulation parfaite adaptée au cas des réseaux fermés de file d'attente qui permet de réaliser une mise à jour en un temps polynomial en le nombre de file et de clients. Cette technique se base sur une représentation compacte des ensembles d'états appelée *diagramme* qui fait l'objet du chapitre 2. Dans le chapitre 4, on adapte la technique présentée dans le chapitre 3 au cas des réseaux fermés multiclasses. La technique de simulation parfaite utilisant les diagrammes est de type enveloppe. Comme cette dernière, un diagramme peut représenter un ensemble d'états plus grand que nécessaire. Le chapitre suivant a pour but la définition des diagrammes. On donnera les propriétés qui font qu'un ensemble d'objets peut être représenté par un diagramme. On verra que l'espace d'états d'un réseau fermé de files d'attente (monoclasse ou multiclasse) satisfait ses propriétés.

# 2 La structure de données *diagramme*

Pour pouvoir effectuer la simulation parfaite de réseaux fermés de files d'attente (monoclasses et multiclasses), l'idée développée dans [10] et [12] est de représenter les ensembles d'états par des graphes appelés *diagrammes*. Ce chapitre a pour but l'introduction de la structure de donnée que sont les diagrammes dans un cadre plus général que celui des réseaux fermés de files d'attente. On verra par exemple que l'ensemble des permutations de $\{1, 2, \ldots, n\}$ peut aussi être représenté par un diagramme. La première section donne les caractéristiques de ce type d'ensemble ainsi que quelques exemples. La seconde section présente les diagrammes et les algorithmes permettant de les manipuler. Ces derniers utilisent le paradigme de programmation dynamique et peuvent être liés à de célèbres algorithmes de programmation dynamique, comme l'algorithme de Held et Karp pour la résolution du problème du voyageur de commerce et l'algorithme de Buzen pour le calcul de la constante de normalisation d'un réseau de Gordon Newell. Les résultats et les définitions de cette seconde section seront ensuite exploités tout au long de ce manuscrit.

## 2.1 La propriété sans mémoire

On s'intéresse aux ensembles qui peuvent êtres représentés par des diagrammes. On les nomme *espace sans mémoire* et ils sont définis à partir d'une suite de fonctions $(f^{(k)})_{k\in\mathbb{N}}$. Soient $K \in \mathbb{N}$ et $M$ appartenant à l'ensemble des images de $f^{(K)}$, un espace sans mémoire $\Omega(K, M)$ est défini comme l'antécédent de $M$ par $f^{(K)}$. Il est composé de vecteurs de dimension $K$ appelés *états*. Soient $\mathbf{x}$ et $\mathbf{y}$ deux états d'un même espace sans mémoire $\Omega(K, M)$, s'il existe $i \leq K$ tel que $f^{(i)}(x_1, \ldots, x_i) = f^{(i)}(y_1, \ldots, y_i)$ alors les vecteurs $(x_1, \ldots, x_i)$ et $(y_1, \ldots, y_i)$ sont interchangeable dans $\mathbf{x}$ et $\mathbf{y}$ au sens où ils n'affectent pas la valeur de $f^{(K)}(\mathbf{x})$ et $f^{(K)}(\mathbf{y})$. Cette propriété sera qualifiée de *sans mémoire*. Ce premier paragraphe est consacré à la définition formelle des espaces sans mémoire.

### 2.1.1 Définitions

**Définition 3.** *Soient $\mathcal{E}$ et $\mathcal{G}$ deux espaces quelconques, $(\mathcal{E}_k)_{k\in\mathbb{N}^*}$ une suite de sous-ensembles de $\mathcal{E}$. On dit que la suite de fonctions $\mathcal{F} := (f^{(k)})_{k\in\mathbb{N}}$ est* ***sans mémoire*** *si :*

*i) la fonction $f^{(0)}$ est constante et a pour valeur $0_{\mathcal{G}} \in \mathcal{G}$ ;*

*ii) pour tout $k > 0$, il existe une fonction $\oplus_k : \mathcal{G} \times \mathcal{E}_k \to \mathcal{G}$ telle que*

$$\begin{array}{rccc} f^{(k)} & : & \mathcal{E}_1 \times \ldots \times \mathcal{E}_k & \to & \mathcal{G} \\ & & (x_1, \ldots, x_{k-1}, x_k) & \mapsto & f^{(k-1)}(x_1, \ldots, x_{k-1}) \oplus_k x_k. \end{array}$$

Pour $k > 0$, $f^{(k)}$ est définie récursivement à partir de $f^{(k-1)}$. La fonction $\oplus_k$ intervient dans chaque définition de $f^{(k)}$, elle peut donc différer pour chaque $k$ comme on le verra dans le chapitre 7. Par la suite, pour simplifier les notations, on écrira $f^{(k)}(\mathbf{x})$ sans se soucier de la taille du vecteur $\mathbf{x}$. De plus, pour un $\mathbf{x}$ fixé de dimension $K \geq k$, $f^{(k)}(\mathbf{x})$ fera référence à $f^{(k)}(x_1, \ldots, x_{k-1}, x_k)$.

Fixons $K \in \mathbb{N}$ et $M \in \mathcal{G}$, on intéresse à l'ensemble des antécédents de $f^{(K)}(\mathbf{x}) = M$.

**Définition 4.** *Soient $M \in \mathcal{E}$ et $K \in \mathbb{N}$ fixés, et $\mathcal{F} = (f^{(k)})_{k\in\mathbb{N}}$ une suite sans mémoire. On appelle **espace sans mémoire** associé aux paramètres $K$ et $M$ l'ensemble des antécédents de $M$ par la fonction $f^{(K)}$. On note $\Omega(K, M)$ cet ensemble, on a ainsi*

$$\Omega(K, M) := {f^{(K)}}^{-1}(M).$$

*Les éléments appartenant à un espace sans mémoire $\Omega(K, M)$ sont appelés **états** de $\Omega(K, M)$.*

Remarquons que si $\mathbf{x}$ et $\mathbf{y}$ sont deux états de $\Omega(K, M)$ tels que qu'il existe $i \leq K$ vérifiant $f^{(i)}(x_1, \ldots, x_i) = f^{(i)}(y_1, \ldots, y_i)$ alors

$$M = f^{(K)}(\mathbf{x}) = f^{(K)}(y_1, \ldots, y_i, x_{i+1}, \ldots, x_K) = f^{(K)}(x_1, \ldots, x_i, y_{i+1}, \ldots, y_K) = f^{(K)}(\mathbf{y}) = M.$$

Donnons maintenant des exemples d'espace sans mémoire.

### 2.1.2 Caractérisation d'une suite sans mémoire à partir d'un monoïde

Un monoïde est un triplet composé d'un ensemble muni d'une loi de composition interne associative et d'un élément neutre. Lorsqu'on dispose d'un monoïde $(\mathcal{E}, \oplus, 0_\mathcal{E})$ on peut définir une suite sans mémoire. Ce type de suite sera utilisé dans les chapitres traitant des réseaux fermés de files d'attente.

**Lemme 2.** *Soit $(\mathcal{E}, \oplus, 0_\mathcal{E})$ un monoïde. Si pour tout $k \in \mathbb{N}$, $\mathcal{E}_k \subseteq \mathcal{E}$ et*

$$\begin{array}{rccc} f^{(k)} & : & \mathcal{E}_1 \times \ldots \times \mathcal{E}_k & \to & \mathcal{E} \\ & & \mathbf{x} & \mapsto & \bigoplus_{j=1}^{k} x_j, \end{array}$$

*alors la suite $\mathcal{F}$ est sans mémoire.*

*Démonstration.* Vérifions que $\mathcal{F}$ est bien sans mémoire. Comme $\mathcal{E} = \mathcal{G}$, pour $k = 0$, la fonction $f^{(0)}$ est constante, sa valeur est $0_\mathcal{E} \in \mathcal{G}$. De plus pour $k > 0$, on a $f^{(k)} : \mathcal{E}_1 \times \ldots \times \mathcal{E}_k \to \mathcal{G}$ et en posant $\oplus_k := \oplus$ on vérifie :

$$f^{(k)}(\mathbf{x}) = \bigoplus_{j=1}^{k} x_j = f^{(k-1)}(\mathbf{x}) \oplus x_k.$$

Les spécifications du lemme 2 s'inscrivent bien dans la définition 3. □

Le lemme 2 impose que $\mathcal{G} = \mathcal{E}$ mais aussi $0_\mathcal{G} = 0_\mathcal{E}$, par contre il ne fixe aucune contrainte concernant les ensembles $\mathcal{E}_k$ si ce n'est $\mathcal{E}_k \subseteq \mathcal{E}$. On donne ci-dessous des exemples de suites sans mémoire caractérisées à partir de monoïdes.

#### Somme cumulée

On s'intéresse aux fonctions calculant les sommes cumulées sur $\mathbb{N}$. Fixons $(\mathcal{E}, \oplus, 0_\mathcal{E}) = (\mathbb{N}, +, 0)$ et pour tout $k \in \mathbb{N}$, $\mathcal{E}_k = \mathbb{N}$ et

$$\begin{array}{rccc} f_{cum}^{(k)} & : & \mathbb{N}^k & \to & \mathbb{N} \\ & & \mathbf{x} & \mapsto & \sum_{i=1}^{k} x_i \end{array}$$

La suite des sommes cumulées $\mathcal{F}_{cum} := (f_{cum}^{(k)})_{k\in\mathbb{N}}$ est sans mémoire. L'espace sans mémoire est donné par

$$\Omega(K,M)_{cum} = \left\{(x_1,x_2,\ldots,x_K)\in\mathbb{N}^K \mid x_1+x_2+\cdots+x_K = M\right\}.$$

Ce dernier correspond à l'espace des états d'un réseau fermé de files d'attente à capacité infinie contenant $K$ files et $M$ clients, comme vu au chapitre 1. Il y a donc

$$|\Omega(K,M)_{cum}| = \binom{K+M-1}{K-1} = \binom{K+M-1}{M} = \frac{(K+M-1)!}{M!(K-1)!}$$

états possibles. En remarquant que $\frac{n^k}{k!} \geq \binom{n}{k} \geq (\frac{n}{k})^k$ et en s'aidant de l'inégalité $k! \geq (\frac{k}{e})^k$ issue de la formule de Stirling [25], on déduit :

$$\left(\frac{M+K-1}{K-1}\right)^{K-1} \leq |\Omega(K,M)_{cum}| \leq \left(\frac{e(M+K-1)}{K-1}\right)^{K-1}. \tag{2.1}$$

Ce type de suite interviendra dans le chapitre 3 dans lequel on réalisera la simulation parfaite de réseaux fermés de files d'attente monoclasses.

#### Somme vectorielle cumulée

On considère ici les sommes cumulées pour des vecteurs d'entiers de dimension $Z$ à valeur dans $\mathbb{N}^Z$. Fixons $(\mathcal{E},\oplus,0_{\mathcal{E}}) = (\mathbb{N}^Z,+,(0,\ldots,0))$, pour tout $k\in\mathbb{N}$, $\mathcal{E}_k = \mathbb{N}^Z$ et

$$\begin{array}{rccc} f_{vec}^{(k)} & : & \mathbb{N}^{Zk} & \to & \mathbb{N}^Z \\ & & \mathbf{x} & \mapsto & \left(\sum_{i=1}^{k} x_{1,i},\ldots,\sum_{i=1}^{k} x_{Z,i}\right). \end{array}$$

La suite $\mathcal{F}_{vec} := (f_{vec}^{(k)})_{k\in\mathbb{N}}$ est sans mémoire. Soit $\mathbf{M} = (M_1,\ldots,M_Z)\in\mathbb{N}^Z$, on a

$$\Omega(K,\mathbf{M})_{vec} = \left\{(x_{z,k})_{z,k}\in\mathbb{N}^{ZK} \;\middle|\; \sum_{i=1}^{k} x_{1,i} = M_1,\ \ldots,\ \sum_{i=1}^{k} x_{Z,i} = M_Z\right\}.$$

On en déduit ainsi que

$$|\Omega(K,\mathbf{M})_{vec}| = \prod_{z=1}^{Z}\binom{K-1}{K+M_z-1}.$$

De plus, en posant $\underline{M} = \min(\{M_z \mid 1\leq z\leq Z\})$ et $\overline{M} = \max(\{M_z \mid 1\leq z\leq Z\})$, on obtient

$$\left(\frac{\underline{M}+K-1}{K-1}\right)^{Z(K-1)} \leq |\Omega(K,\mathbf{M})_{vec}| \leq \left(\frac{e(\overline{M}+K-1)}{K-1}\right)^{Z(K-1)}.$$

Ce type de suite interviendra dans le chapitre 4 dans lequel on réalise la simulation parfaite des réseaux fermés de files d'attente multiclasses.

#### Permutation

Une autre loi de composition associative est l'union d'ensembles, prenons $(\mathcal{E}, \oplus, 0_{\mathcal{E}}) = (\mathcal{P}(\mathbb{N}), \cup, \emptyset)$ et pour tout $k \in \mathbb{N}$ :

$$\begin{array}{rccc} f_{per}^{(k)} & : & \mathcal{P}(\mathbb{N})^k & \to & \mathcal{P}(\mathbb{N}) \\ & & \mathbf{x} & \mapsto & \bigcup_{i=1}^{k} x_i. \end{array}$$

La suite $\mathcal{F}_{per} = (f_{per}^{(k)})_{k\in\mathbb{N}}$ est sans mémoire.

Considérons maintenant que pour chaque $k \in \mathbb{N}$, $\mathcal{E}_k$ est l'ensemble des singletons de $\mathcal{P}(\mathbb{N})$. Pour $\mathcal{M} = \{1, 2, \ldots, K\}$ on a :

$$\Omega(K, \mathcal{M})_{per} = \left\{(x_1, x_2, \ldots, x_K) \in \mathcal{P}(\mathbb{N})^K \;\middle|\; x_1 \cup x_2 \cup \ldots \cup x_K = \mathcal{M}\right\}.$$

L'espace sans mémoire $\Omega(K, \mathcal{M})_{per}$ est donc en bijection avec l'ensemble des permutations de taille $K$. Ainsi $|\Omega(K, \mathcal{M})_{per}| = K!$.

#### Maximum

Considérons la suite des maxima, fixons $(\mathcal{E}, \oplus, 0_{\mathcal{E}}) = (\mathbb{N}, \max, 0)$. Pour tout $k \in \mathbb{N}$ on pose donc :

$$\begin{array}{rccc} f_{max}^{(k)} & : & \mathbb{N}^k & \to & \mathbb{N} \\ & & \mathbf{x} & \mapsto & \max(x_1, \ldots, x_k) \end{array}$$

La suite $\mathcal{F}_{max} = (f_{max}^{(k)})_{k\in\mathbb{N}}$ est sans mémoire. Considérons pour tout $k > 0$, $\mathcal{E}_k = \mathbb{N}$, on a alors

$$\Omega(K, M)_{max} = \left\{(x_1, x_2, \ldots, x_K) \in \mathbb{N}^K \;\middle|\; \forall k \leq K,\ x_k \leq M \text{ et } \exists\, i \leq K,\ x_i = M\right\},$$

et $|\Omega(K, M)_{max}| = (M+1)^K - M^K$.

### 2.1.3 Simulation d'une chaîne de Markov à horizon fixé

Au chapitre 1, nous avons vu comment simuler une chaîne de Markov à valeurs dans un espace d'états fini. Soit $(X_n)_{n\in\mathbb{N}}$ une chaîne à valeur dans $\mathcal{S}$ fini. On suppose que l'on dispose d'une suite d'innovations $(U_n)_{n\in\mathbb{N}}$ (de variables aléatoires) i.i.d. qui suivent une loi uniforme sur $[0, 1[$ et d'une fonction de mise à jour $\mathrm{maj} : \mathcal{S} \times [0, 1[ \to \mathcal{S}$. Pour simuler $(X_n)_{n\in\mathbb{N}}$ du temps 0 jusqu'au temps $K$, on commence par choisir $s_0 \in \mathcal{S}$ puis on effectue les mises à jour à l'aide de la fonction $\mathrm{maj}$, autrement dit :

$$\begin{cases} X_0 = s_0 \\ X_n = \mathrm{maj}(X_{n-1}, U_n) \quad \text{pour } 1 \leq n \leq K. \end{cases}$$

On peut traduire ce procédé par une suite sans mémoire en reconsidérant la suite de variables aléatoires. En effet, soient $\mathcal{E} = [0,1[$, $\mathcal{G} = \mathcal{S}$ et pour tous $k \in \mathbb{N}$, $\mathcal{E}_k = \mathcal{E}$. Considérons la suite $\mathcal{F}_{mc} := (f_{mc}^{(k)})_{k\in\mathbb{N}}$ définie par :

$$f_{mc}^{(k)}(\mathbf{u}) = \begin{cases} s_0 & \text{si } k = 0 \\ \text{maj}(f_{mc}^{(k-1)}(\mathbf{u}), u_k) & \text{sinon.} \end{cases}$$

On a $0_{\mathcal{G}} = s_0$ et pour tout $k > 0$ la fonction $\oplus$ telle que $s \oplus u := \text{maj}(s,u)$. La suite $\mathcal{F}_{mc}$ s'inscrit bien dans le cadre de la définition 3, elle est donc sans mémoire. Maintenant, fixons $K \in \mathbb{N}$ et $M \in \mathcal{S}$, on a

$$\Omega(K,M)_{mc} = \left\{(u_1,\dots,u_K) \in [0,1[^K \mid X_0 = s_0 \text{ et } X_K = M\right\}.$$

L'ensemble $\Omega(K,M)_{mc}$ correspond donc à l'ensemble de tous les vecteurs $\mathbf{u} := (u_1,\dots,u_K) \in [0,1[^K$ tels que

$$\begin{cases} X_0 = s_0 & \text{si } k = 0 \\ X_k = \text{maj}(X_{k-1}, u_k) & \text{pour } 1 \le k \le K, \end{cases}$$

et pour lesquels $X_K = M$.

L'ensemble $\Omega(K,M)_{mc}$ correspond donc à l'ensemble de toutes les suites d'innovations pour lesquelles en fixant $X_0 = s_0$ et la fonction de mise à jour maj, on a $X_K = M$.

**Exemple 7** (Simulation d'une file d'attente)**.** *Prenons pour exemple une file d'attente $M/M/2/10$. Supposons que les deux serveurs ont le même taux de service. On le note $a$. Le taux d'arrivée dans la file est noté $c$, il est tel que $c < 2a$ avec $2a + c = 1$. La chaîne de Markov $(X_n)_{n\in\mathbb{N}}$ modélisant le nombre de clients de cette file prend ses valeurs dans $\mathbb{N}$ et est définie telle que :*

$$\text{maj}(s,u) = \begin{cases} \max(s-1,0) & \textit{si } u \in [0,a[ \\ s & \textit{si } s < 2 \textit{ et } u \in ]a,2a] \textit{ (le serveur sert une file vide)} \\ s-1 & \textit{si } s \ge 2 \textit{ et } u \in ]a,2a] \\ \min(s+1,10) & \textit{sinon (arrivée dans la file).} \end{cases}$$

*Supposons qu'à l'instant $0$ on ait $X_0 = 0$. La figure 2.1 illustre les différentes trajectoires pour parvenir à $X_5 = 0$ en partant de $X_0 = 0$. On en dénombre $22$. Parmi ses $22$ trajectoires, on a notamment la suivante $X_1 = 1$, $X_2 = 1$, $X_3 = 2$, $X_4 = 1$ et $X_5 = 0$. Elle est illustrée en gras dans la figure. Cette trajectoire est obtenue à partir de vecteurs $\mathbf{u} \in \Omega(5,0)$ tels que $u_1 \in [2a,1[$, $u_2 \in [a,2a[$, $u_3 \in [2a,1[$, $u_4 \in [0,a[\cup[a,2a[ = [0,2a[$ et $u_5 \in [0,a[$. La figure 2.1 illustre ce que nous appellerons dans la section suivante un diagramme.*

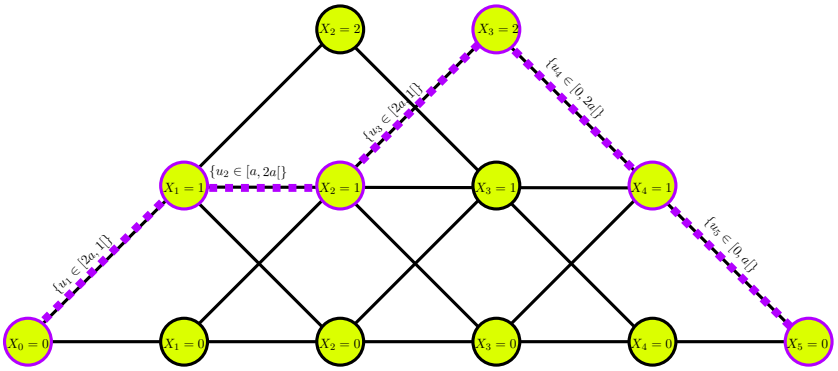



**FIGURE 2.1:** Trajectoires telles que $X_0 = 0$ et $X_5 = 0$.

## 2.2 Les diagrammes

L'objectif de cette section est la présentation des diagrammes, de leurs propriétés et des algorithmes qui permettent de les traiter. Un diagramme est un graphe $D = (\mathcal{N}, A)$ qui représente un ensemble d'états $S \subseteq \Omega(K, M)$.

On considère dans cette section $\mathcal{F}$ une suite sans mémoire, $K \in \mathbb{N}$, $M \in \mathcal{E}_K$ et $\Omega(K, M)$ est non vide. De plus on fait pour hypothèse que $\mathcal{E}$ est un ensemble discret et que $\Omega(K, M)$ est un ensemble fini. Tous les exemples que nous venons de voir s'inscrivent dans ce cadre hormis celui de la simulation d'une chaîne de Markov à horizon fixé. On reviendra à ce dernier à la fin de cette section.

### 2.2.1 Définition

Commençons par les nœuds du graphe. On définit la fonction $h$ qui à un état associe un ensemble de $(K+1)$ couples appartenant à l'ensemble $\{0, 1, \ldots, K\} \times \mathcal{G}$.

$$\begin{array}{rcccl} h & : & \Omega(K,M) & \to & \mathcal{P}(\{0,1,\ldots,K\} \times \mathcal{G}) \\ & & \mathbf{x} & \mapsto & \{(k, f^{(k)}(\mathbf{x})) \mid k \in \{0,\ldots,K\}\} \end{array}$$

L'ensemble de nœuds est $\mathcal{N} := \bigcup_{\mathbf{x}\in\Omega(K,M)} h(\mathbf{x})$.

Continuons avec les arcs. On définit la fonction $g$ qui à un état associe un ensemble d'arcs.

$$\begin{array}{rcccl} g & : & \Omega(K,M) & \to & \mathcal{P}(\mathcal{N} \times \mathcal{N}) \\ & & \mathbf{x} & \mapsto & \bigcup_{i=1}^{K} \Big\{ \big((i-1, f^{(i-1)}(\mathbf{x})), (i, f^{(i)}(\mathbf{x}))\big) \Big\}. \end{array}$$

**Définition 5.** *On dit que le graphe orienté $D = (\mathcal{N}, A)$ est un **diagramme** s'il existe un ensemble d'états $S \subseteq \Omega(K, M)$ non vide tel que :*

$i)$ $\mathcal{N} = \bigcup_{\mathbf{x}\in\Omega(K,M)} h(\mathbf{x})$,

$ii)$ $A = g(S) := \bigcup_{\mathbf{x}\in S} g(\mathbf{x})$.

La figure 2.2 illustre un diagramme obtenu à partir d'un ensemble contenant deux états.

Par définition, l'ensemble $\mathcal{N}$ des nœuds est commun à tous les diagrammes. Considérons maintenant un diagramme $D = (\mathcal{N}, A)$ obtenu à partir d'un ensemble d'états $S \subseteq \Omega(K, M)$. Soit $\mathbf{x} \in S$, $g(\mathbf{x})$ est un ensemble d'arcs. Il peut être vu graphiquement comme un chemin débutant au nœud $(0, f^{(0)}(\mathbf{x}))$ et se terminant au nœud $(K, f^{(K)}(\mathbf{x}))$. Or pour tout $\mathbf{x} \in \Omega(K, M)$, on a $f^{(0)}(\mathbf{x}) = 0_{\mathcal{G}}$ et $f^{(K)}(\mathbf{x}) = M$. Les nœuds $(0, 0_{\mathcal{G}})$ et $(K, M)$ sont donc communs à tous les chemins $g(\mathbf{x})$. On les nommera respectivement **nœud source** et **nœud destination**. On désignera par **chemin** un ensemble de $K$ arcs reliant les nœuds source et destination.

Notons $A(k) := \{\big((k-1, \ell), (k, \ell')\big) \in A\}$, alors cet ensemble d'arc correspond graphiquement à l'ensemble des arcs contenus dans la $k$-ième colonne du diagramme $D = (\mathcal{N}, A)$. Les ensembles $A(k)$ sont disjoints deux à deux et $A = \cup_{k=1}^{K} A(k)$, ainsi le nombre d'arcs dans un diagramme est donné par

$$|A| = \sum_{k=1}^{K} |A(k)|.$$

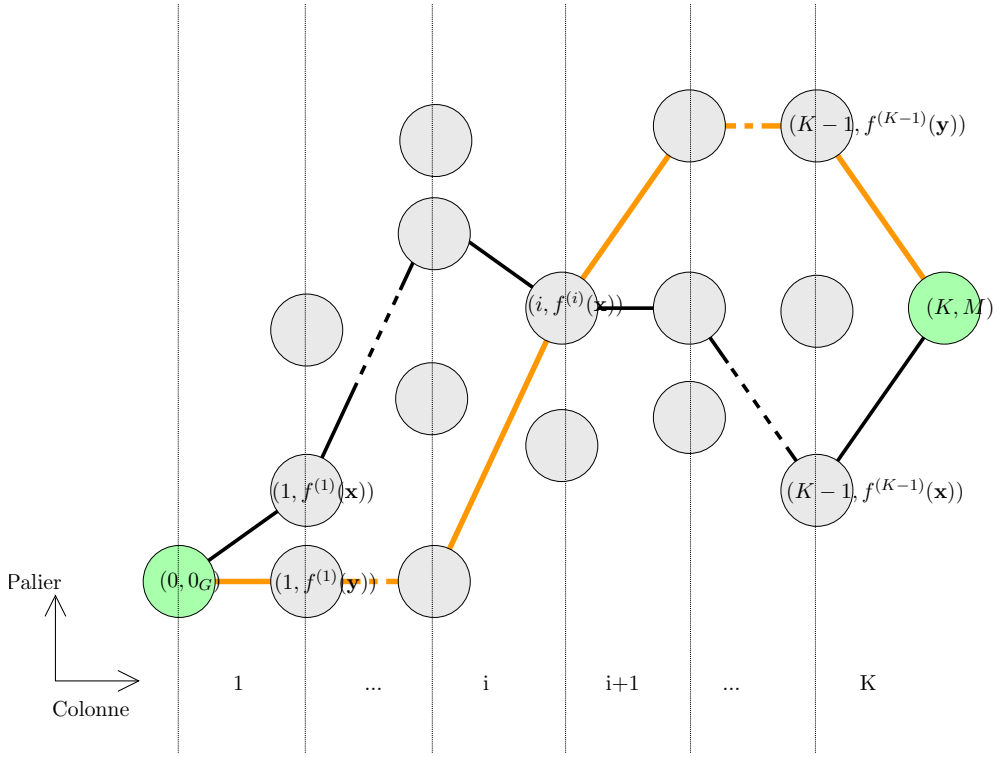



**FIGURE 2.2:** Diagramme pour $S = \{\mathbf{x}, \mathbf{y}\}$.

Si $S = \Omega(K, M)$, on dit que le diagramme est **complet**. On le note $\mathcal{D}(K, M)$ ou simplement $\mathcal{D}$ pour alléger les notations. L'ensemble de ses arcs est noté $\mathcal{A}$. Ainsi $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ avec $\mathcal{A} = g(\Omega(K, M))$. On définit la **complexité** de la représentation par diagramme comme étant le nombre d'arcs du diagramme complet, c'est-à-dire $|\mathcal{A}|$.

Étant donnés deux diagrammes $D_1 = (\mathcal{N}, A_1)$ et $D_2 = (\mathcal{N}, A_2)$, on dit que $D_1$ est un **sous-diagramme** de $D_2$ si $A_1 \subseteq A_2$. On note alors $D_1 \subseteq D_2$. Pour tout diagramme $D = (\mathcal{N}, A)$, il existe un ensemble d'états $S \subseteq \Omega(K, M)$ tel que $g(S) = A$. Comme la fonction $g$ qui associe à un ensemble d'état un ensemble d'arcs est monotone (au sens de l'inclusion d'ensembles), l'ensemble $A$ est toujours inclus dans l'ensemble des arcs du diagramme complet autrement dit $A = g(S) \subseteq g(\Omega(K, M)) = \mathcal{A}$. Remarquons alors que chaque diagramme est donc un sous-diagramme du diagramme complet. On notera par la suite, $D \subseteq \mathcal{D}(K, M)$ pour préciser que $D$ est un diagramme construit à partir d'un ensemble d'état appartenant à $\Omega(K, M)$.

## 2.2.2 Exemples

On illustre maintenant les diagrammes des exemples *somme cumulée* et *permutation* de la première section.

#### Diagramme des sommes cumulées

La suite des sommes cumulées $\mathcal{F}_{cum}$ est définie telle que $f^{(k)}_{cum}(\mathbf{x}) = \sum_{i=0}^{k} x_i$. Considérons l'espace des états $\Omega(5, 3)_{cum}$ et les ensembles $S_1 = \{(2, 0, 0, 0, 1), (1, 0, 1, 1, 0)\}$ et $S_2 = \Omega(5, 3)_{cum}$. Par définition, $D_1 = (\mathcal{N}, g(S_1))$ et $D_2 = (\mathcal{N}, g(S_2))$ sont des diagrammes. De plus comme

$S_2 = \Omega(5,3)_{cum}$, le diagramme $D_2$ est complet. La figure suivante illustre les diagrammes $D_1$ et $D_2$.

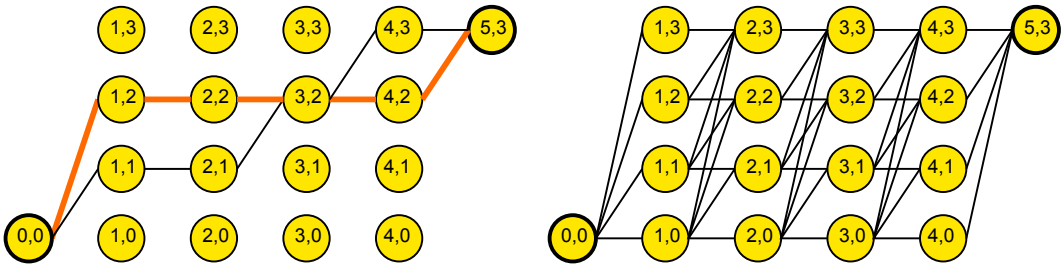



**FIGURE 2.3:** Diagrammes $D_1$ (à gauche) et $D_2$ (à droite).

Dans $D_1$, le chemin $g(\mathbf{x})$ pour $\mathbf{x} = (2,0,0,0,1)$ correspond au trait gras et orange. Il atteint successivement les paliers $2,2,2,2,3$ ce qui correspond au vecteur somme cumulée de $\mathbf{x}$. Le diagramme complet de la figure 2.3 contient $38$ arcs, en effet $|A(1)| = |A(5)| = 4$ et $|A(2)| = |A(3)| = |A(4)| = 10$.

**Lemme 3.** *Le diagramme complet des sommes cumulées $\mathcal{D}_{cum}$ contient $(K-2)\frac{(M+1)(M+2)}{2} + 2(M+1)$ arcs.*

*Démonstration.* Soit $\mathcal{D}_{cum}(K,M) = (\mathcal{N},\mathcal{A})$ le diagramme complet des sommes cumulées. Chaque colonne $k$ pour $1 < k < K$ contient $M+1$ nœuds de la forme $(k,\ell)$ qui correspondent aux $M+1$ valeurs que peut prendre $f^{(k)}_{cum}(\mathbf{x})$ pour $\mathbf{x} \in \Omega(K,M)_{cum}$. Chaque nœud $(k,\ell)$ a $M+1-\ell$ arcs sortants. Comme $\ell \in \{0,1,\ldots,M\}$, on a

$$|A(k)| = (M+1) + M + \ldots + 1 = \frac{(M+1)(M+2)}{2}.$$

Il n'y a qu'un seul nœud source et qu'un seul nœud destination, les colonnes $1$ et $K$ contiennent $M+1$ arcs. Ainsi on en déduit que :

$$|\mathcal{A}| = \sum_{k=1}^{K} |A(k)| = (K-2)\frac{(M+1)(M+2)}{2} + 2(M+1).$$

□

La complexité de la représentation d'un diagramme des sommes cumulées est en $O(KM^2)$.

### Diagramme des permutations

Considérons la suite sans mémoire $\mathcal{F}_{per}$. La figure 2.4 illustre le diagramme complet pour $\mathcal{M} = \{1,2,3,4\}$. Tous les chemins du nœud $(0,\emptyset)$ au nœud $(4,\mathcal{M})$ du diagramme correspondent à des permutations. La permutation

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 3 & 1 \end{pmatrix}$$

est en bijection avec l'état $(\{2\},\{4\},\{3\},\{1\}) \in \Omega(K,\mathcal{M})_{per}$ qui correspond lui-même au chemin orange en pointillés dans le diagramme de la figure 2.4.

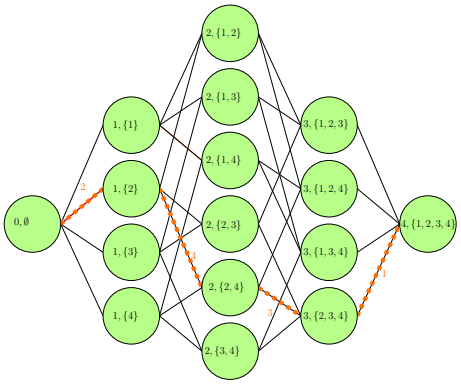



**FIGURE 2.4:** Diagramme des permutations de $\{1, 2, 3, 4\}$.

Le diagramme complet de la figure 2.4 contient 32 arcs ($|A(1)| = |A(4)| = 4$ et $|A(2)| = |A(3)| = 12$). L'espace des états $\Omega(4, \{1, 2, 3, 4\})_{per}$ contient 24 états. Remarquons de plus que le diagramme complet correspond au diagramme de Hasse [7] pour la relation d'ordre $\subseteq$ sur l'ensemble des parties de l'ensemble $\{1, 2, 3, 4\}$.

**Lemme 4.** *Le diagramme complet des permutations $\mathcal{D}_{per}$ contient $K2^{K-1}$ arcs.*

*Démonstration.* L'ensemble des nœuds du diagramme $\mathcal{D}_{per} = (\mathcal{N}, \mathcal{A})$ contient autant de nœuds de la forme $(k, \{m_1, \ldots, m_k\})$ qu'il y a de façons de choisir $k$ éléments parmi $K$, c'est-à-dire $\binom{K}{k}$ nœuds. Un nœud $(k-1, \{m_1, \ldots, m_{k-1}\}) \in \mathcal{N}$ tel que $1 \leq k \leq K$ a pour arcs sortants :

$$\{\big((k-1, \{m_1, \ldots, m_{k-1}\}), (k, \{m_1, \ldots, m_{k-1}, m\})\big) \mid m \notin \{m_1, \ldots, m_{k-1}\} \text{ et } 1 \leq m \leq K\},$$

et possède donc $K - k + 1$ arcs sortants. Chaque colonne $1 \leq k \leq K$ possède donc $|\mathcal{A}(k)| = (K - k + 1)\binom{K}{k-1}$ arcs. On en déduit ainsi que

$$|\mathcal{A}| = \sum_{k=1}^{K} |\mathcal{A}(k)| = \sum_{k=0}^{K-1} (K-k)\binom{K}{k} = K \sum_{k=0}^{K-1} \binom{K-1}{k} = K2^{K-1}.$$

□

La complexité de la représentation par diagramme permutation est en $O(K2^K)$.

### 2.2.3 Fonctions de transformation et propriétés

Intéressons-nous maintenant aux fonctions qui permettent de transformer un ensemble d'états en un diagramme et inversement. Pour un diagramme $D = (\mathcal{N}, A) \subseteq \mathcal{D}(K, M)$ et un ensemble d'états $S \subseteq \Omega(K, M)$, on définit

$$\phi(S) := (\mathcal{N}, g(S)) \qquad \text{et} \qquad \psi(D) := \bigcup_{S \subseteq \Omega(K,M) \text{ tel que } A = g(S)} S.$$

La fonction $\psi$ transforme le diagramme en le plus grand sous-ensemble d'états tel que $g(S) = A$.

**Exemple 8** (Somme cumulée)**.** *Reprenons le diagramme $D_1$ de la figure 2.3. On a $S_1 = \{(2, 0, 0, 0, 1), (1, 0, 1, 1, 0)\}$ et $D_1 = (\mathcal{N}, g(S_1))$. Graphiquement, le chemin orange et le chemin noir se croisent au nœud $(3, 2)$. Ainsi si on considère $S_3 = \{(2, 0, 0, 1, 0), (1, 0, 1, 0, 1)\}$*

*alors $g(S_1) = g(S_3)$. On peut alors définir le diagramme $D_1$ à partir de l'ensemble $S_3$ par $D_1 = (\mathcal{N}, g(S_3))$. Si on regarde tous les chemins possibles de $D_1$, il y en a 4 (le orange, le noir, le orange-noir et le noir-orange) : ils correspondent à l'ensemble $S_4 = S_1 \cup S_3$. Comme $S_4$ est le plus grand ensemble d'états tel que $D_1 = (\mathcal{N}, g(S_4))$, on a donc $\psi(D_1) = S_4$.*

**Lemme 5.** *Pour tout ensemble d'états $S \subseteq \Omega(K, M)$ et tout diagramme $D = (\mathcal{N}, A) \subseteq \mathcal{D}(K, M)$, les propriétés suivantes sont vérifiées :*

*i)* $S \subseteq \psi \circ \phi(S)$,

*ii)* $D = \phi \circ \psi(D)$.

Considérons comme relations d'ordre, l'inclusion d'ensemble pour les états et l'inclusion d'arcs pour les diagrammes. $ii)$ implique que $\forall S,\ \phi(S) = \phi \circ \psi \circ \phi(S)$ et $\forall D,\ \psi(D) = \psi \circ \phi \circ \psi(D)$, ainsi $(\psi, \phi)$ est une correspondance de Galois qui est isotone car $S \subseteq \psi \circ \phi(S)$.

*Démonstration.* $i)$ Posons $D = \phi(S)$. Comme $D$ est un diagramme tel que l'ensemble de ses arcs est $g(S)$, on a par définition de la fonction $\psi$, $S \subseteq \psi(D) = \psi \circ \phi(S)$.
$ii)$ Posons $S = \psi(D)$, alors $g(S) = A$ et donc $D = (\mathcal{N}, g(S)) = \phi \circ \psi(D)$. □

### 2.2.4 Valuation d'un arc

Chaque arc $\big((k-1, m), (k, m')\big)$ d'un diagramme $D$ encode l'information « il existe $\mathbf{x} \in \psi(D)$ tel que $f^{(k-1)}(\mathbf{x}) = m$ et $f^{(k)}(\mathbf{x}) = m'$».

Une fonction de valuation permet de retrouver pour $m$ et $m'$ fixés, l'ensemble des $x_k$ encodés par un arc, c'est-à-dire les $x_k$ vérifiant $f^{(k-1)}(x_1, \ldots, x_{k-1}) = m$ et $f^{(k)}(x_1, \ldots, x_{k-1}, x_k) = m'$.

#### Valuation

On appelle **fonction de valuation**, la fonction qui à chaque arc associe un sous ensemble de $\mathcal{E}$. Elle est définie par :

$$\begin{array}{rcccl} v & : & A & \to & \mathcal{P}(\mathcal{E}) \\ & & \big((k-1, m), (k, m')\big) & \mapsto & \left\{x_k \in \mathcal{E}_k \mid \mathbf{x} \in \Omega(K, M),\ f^{(k-1)}(\mathbf{x}) = m \text{ et } f^{(k)}(\mathbf{x}) = m'\right\}. \end{array}$$

Remarquons que si $a \in A(k)$ alors $v(a) \subseteq \mathcal{E}_k$.

Dans un diagramme des sommes cumulées, la valeur d'un arc est un singleton. En effet, pour tout arc $a = \big((k-1, m), (k, m')\big) \in A$ $(m < m')$, $v(a) = \{m' - m\}$ et donc $|v(a)| = 1$. La valeur d'un arc dans un diagramme des permutations est également un singleton. Dans un diagramme des permutations, un arc $a = \big((k-1, m), (k, m')\big) \in A$ à pour valeur le singleton (de singleton) $v(a) = \{m' \setminus m\}$ et donc $|v(a)| = 1$.

Si pour tout $a \in \mathcal{A}$ du diagramme complet, on a $|v(a)| = 1$ alors pour tout diagramme $D \subseteq \mathcal{D}$ on aura également $|v(a)| = 1$. Chaque arc est donc associé à une seule valeur $x_k$, ce qui implique que chaque chemin dans le diagramme est associé à un seul état. La propriété $ii)$ du lemme 5 implique que si $S \in \Omega(K, M)$ est tel que $S = \psi(D)$ alors l'inclusion de $i)$ est une égalité. Le corollaire ci-dessous utilise cette remarque sur deux cas pour lesquels on a $S = \psi(D)$ : lorsque le diagramme est complet et lorsque le diagramme ne contient qu'un seul chemin.

**Corollaire 1.** *Soit $D \subseteq \mathcal{D}(K, M) := (\mathcal{N}, \mathcal{A})$ un diagramme tel que pour tout $a \in \mathcal{A}$, $|v(a)| = 1$ alors*

*i) $D$ est complet si et seulement si $\psi(D) = \Omega(K, M)$,*

*ii) $D$ ne contient qu'un chemin (i.e. $|A| = K$) si et seulement si $|\psi(D)| = 1$.*

Ce corollaire interviendra lorsqu'on utilisera des diagrammes dans le cadre de la simulation parfaite. Notamment pour justifier que si $D$ ne contient qu'un seul chemin, alors $D$ est en bijection avec un ensemble d'états ne contenant qu'un seul état. À la fin de la simulation, on pourra alors extraire cet état à l'aide de la fonction $\psi$. Pour alléger les notations, si les diagrammes induits par une suite sans mémoire $\mathcal{F}$ sont tels que $|v(a)| = 1$ pour tour arc, on écrira alors $v(a) = m$ à la place de $v(a) = \{m\}$.

Intéressons-nous maintenant à la suite $\mathcal{F}_{max}$ qui induit des diagrammes pour lequel il existe des arcs tels que $v(a) > 1$.

#### Diagramme des maxima

Pour les mêmes paramètres $K \in \mathbb{N}$ et $M \in \mathbb{N}$, le diagramme complet des maxima $\mathcal{D}_{max}$ est graphiquement identique au diagramme complet des sommes cumulées $\mathcal{D}_{cum}$. En effet, dans les deux cas on a $\mathcal{E} = \mathcal{E}_k = \mathbb{N}$. De plus $h(\Omega(K, M)_{max}) = h(\Omega(K, M)_{cum})$ et $g(\Omega(K, M)_{max}) = g(\Omega(K, M)_{cum})$ ce qui implique que $\mathcal{N}_{max} = \mathcal{N}_{cum}$ et $\mathcal{A}_{max} = \mathcal{A}_{cum}$. Seules les fonctions de valuations $v_{max}$ et $v_{cum}$ diffèrent. En effet, pour $a = \big((k-1, \ell), (k, \ell')\big)$ un arc appartenant aux deux diagrammes complets, on a toujours $\ell \leq \ell'$ mais $v_{cum}(a) = \ell' - \ell$ et

$$v_{max}(a) = \begin{cases} \{m \mid 0 \leq m \leq \ell\} & \text{si } \ell = \ell' \\ \{\ell'\} & \text{sinon.} \end{cases}$$

La figure 2.3 illustre également un diagramme pour le cas des maxima. En revanche, comme $|v_{max}(a)| \geq |v_{cum}(a)|$, un diagramme des maxima représente au moins autant d'états qu'un diagramme des sommes cumulées ayant les mêmes arcs.

### 2.2.5 Chemins dans un diagramme

Nous avons vu qu'à chaque état $\mathbf{x} \in S$ est associé un ensemble d'arcs $g(\mathbf{x})$. L'ensemble des arcs $g(\mathbf{x})$ correspond à un chemin dans le diagramme $D$. On définit $path(a)$, l'ensemble des arcs pouvant être reliés à l'arc $a \in A$ par un chemin dans le diagramme $D$.

$$path(a) = \bigcup_{\{\mathbf{x} \in \psi(D) \mid a \in g(\mathbf{x})\}} g(\mathbf{x}),$$

et $\mathcal{P}ath(A')$ l'ensemble des arcs pouvant être reliés à un ensemble d'arcs $A' \subseteq A$ :

$$\mathcal{P}ath(A') = \bigcup_{a \in A'} path(a).$$

Ces définitions seront utilisées pour manipuler les ensembles $S' \subseteq \psi(D)$ dont les éléments ont une contrainte particulière concernant une ou plusieurs de leurs composantes, par exemple pour $e \in \mathcal{E}_k$ fixé, l' ensemble $S' = \{\mathbf{x} \in \psi(D) \mid x_k = e\}$.

### 2.2.6 Algorithmes

On présente maintenant les algorithmes relatifs aux diagrammes, on commence par expliquer comment transformer un ensemble d'états en un diagramme. Les algorithmes `CardStates`, `DiagramToStates` et `RandState` utilisent le paradigme de programmation dynamique introduit par Bellman en 1950 [6] et faisant l'objet du chapitre 15 du livre [22]. On donnera des exemples d'applications de ces algorithmes.

L'algorithme `CardStates` permet de calculer le nombre d'états représentés par un diagramme. Dans le chapitre 3, les diagrammes seront utilisés comme enveloppe pour la simulation parfaite et `CardStates` permettra de connaître à l'évolution du nombre d'états au cours de la simulation. L'algorithme `DiagramToStates` permet de transformer un diagramme en l'ensemble d'états qu'il représente, il sera utilisé au chapitre 3 pour réaliser la technique *splitting* [18]. L'algorithme `RandState` effectue une marche aléatoire dans le diagramme selon la loi donnée dans la formule (2.5). Sa marche commence au nœud source et se termine au nœud destination. Il permet l'échantillonnage aléatoire d'états représentés par le diagramme et sera utilisé dans les chapitres 6 et 7 comme un générateur de Boltzmann.

#### Conversion d'un ensemble d'états en un diagramme

Soit $S \subseteq \Omega(K, M)$ un ensemble d'états. Grâce à la fonction $g$, on peut facilement déduire un algorithme pour construire le diagramme $D = \phi(S)$. On commence par initialiser l'ensemble $A$ des arcs de $D$ à l'ensemble vide. Puis pour chaque état $\mathbf{x} \in S$ on ajoute à $A$ l'ensemble des arcs correspondant à $g(\mathbf{x})$.

Si $S = \Omega(K, M)$, cet algorithme est très mauvais car il contraint à considérer tous les états possibles. Sa complexité en temps de calcul correspond à la cardinalité de l'espace sans mémoire, c'est-à-dire en $O(|\Omega(K, M)|)$. Pour encoder un diagramme complet $\mathcal{D}(K, M) := (\mathcal{N}, \mathcal{A})$, mieux vaut donc exploiter la structure du diagramme. Par exemple, le diagramme complet des sommes cumulées peut être construit avec une complexité en $O(KM^2)$. En effet, il suffit de construire l'ensemble des arcs, colonne par colonne : dans la première et la dernière colonne on place $M + 1$ arcs et dans les colonnes intermédiaires on en place $\frac{(M+1)(M+2)}{2}$. On obtient ainsi $\mathcal{A} = \cup_{1 \leq k \leq K} \mathcal{A}(k)$ avec

- $\mathcal{A}(1) = \{\big((0, 0), (1, \ell)\big),\ 0 \leq \ell \leq M\}$
- pour tout $k > 0$ et $k < K$, $\mathcal{A}(k) = \{\big((k-1, \ell), (k, \ell')\big),\ 0 \leq \ell \leq \ell' \leq M\}$
- $\mathcal{A}(K) = \{\big((K-1, \ell), (K, M)\big),\ 0 \leq \ell \leq M\}$.

#### Calcul du nombre d'états dans un diagramme

L'algorithme `CardStates` détermine le nombre d'états représentés par un diagramme, autrement dit pour $D$ un diagramme, il retourne $|\psi(D)|$. L'idée est d'attribuer à chaque nœud $n = (k, m)$ du diagramme une valeur notée $Card(n)$ qui compte le nombre de préfixes $(x_1, \ldots, x_k)$ de longueur $k$ tels que $f^{(k)}(\mathbf{x}) = m$ et $\mathbf{x} \in \psi(D)$. Pour chaque nœud différent du nœud source, on initialise $Card(n)$ à 0. Le nœud source prend pour valeur 1. On traite les colonnes les unes après les autres dans l'ordre croissant. On attribue au nœud $n = (k, m)$, la somme des valeurs de tous les nœuds $(k-1, \ell)$ tels que $a = \big((k-1, \ell), (k, m)\big) \in A$ multiplié par le cardinal de $v(a)$ (ligne 6). À la fin de l'algorithme, le nœud destination a pour valeur $Card((K, M)) = |\psi(D)|$. Il y a autant d'additions et de multiplications à faire qu'il y a d'arcs dans le diagramme, la complexité en temps de calcul de cet algorithme est donc linéaire en le nombre d'arcs, c'est-à-dire en $O(|A|)$.

**Algorithme 5:** `CardStates`

**Données** : $D = (\mathcal{N}, A) \subseteq \mathcal{D}(K, M)$, $v$ (la fonction de valuation).
**Résultat** : $|\psi(D)| \in \mathbb{N}$

```
pour chaque n ∈ 𝒩 faire
    Card(n) ← 0;
Card((0, 0_𝒢)) ← 1;
pour k = 1, 2, ..., K faire
    pour chaque a = ((k − 1, ℓ), (k, m)) ∈ A(k) faire
        Card((k, m)) ← Card((k, m)) + Card((k − 1, ℓ))|v(a)|;
renvoyer Card((K, M)).
```

**Exemple 9.** *Le diagramme de la figure 2.5 représente les* $\frac{4!}{4} = 6$ *permutations de l'ensemble* $\{1, 2, 3, 4\}$ *qui débutent par le chiffre* 1. *Dans un diagramme des permutations, chaque arc encode un unique élément (*$|v(a)| = 1$*). L'algorithme* `CardStates` *appliqué à ce diagramme, attribue à chaque nœud du diagramme une valeur et retourne celle du nœud final (*$(4, \{1, 2, 3, 4\})$*). Cette dernière correspond bien à la valeur attendue* 6.

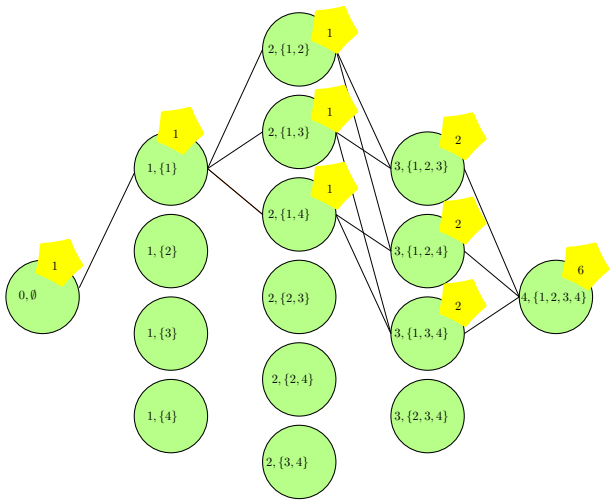



**FIGURE 2.5:** Diagramme des permutations de $\{1, 2, 3, 4\}$ débutant par 1.

**Lemme 6.** *Soit* $D \subseteq \mathcal{D}(K, M)$ *un diagramme, alors* `CardStates(D)` *retourne la valeur* $|\psi(D)|$.

*Démonstration.* Posons $S = \psi(D)$, on a alors $D = (\mathcal{N}, g(S))$. On veut démontrer que l'algorithme retourne $|S|$. Pour $m \in \mathcal{D}$, notons $S_{(k,m)}$ l'ensemble des préfixes des éléments de $S$ de longueur $k$ avec $0 \leq k \leq K$ tels que

$$S_{(k,m)} := \left\{ (x_1, \ldots, x_k) \mid f^{(k)}(\mathbf{x}) = m \text{ et } \mathbf{x} \in S \right\}.$$

On procède par récurrence sur $k$ en montrant que pour tout nœud du diagramme $(k, m) \in \mathcal{N}$, on a

$$Card((k, m)) = |S_{(k,m)}|. \tag{2.2}$$

**Cas** $k = 0$. On a $m = 0_{\mathcal{G}}$ et $S_{(0,0_{\mathcal{G}})} = \{\epsilon\}$ ($\epsilon$ est le vecteur de dimension 0). Ainsi on a bien $|S_{(0,0_{\mathcal{G}})}| = 1 = Card((0, 0_{\mathcal{G}}))$.

**Cas $k \geq 1$.** Remarquons que :

$$\begin{aligned} S_{(k+1,m)} &= \Big\{(x_1,\dots,x_k,x_{k+1}) \mid \mathbf{x} \in S \text{ et } f^{(k+1)}(\mathbf{x}) = m\Big\} \\ &= \bigcup_{\{a\in A \mid a=\left((k,m'),(k+1,m)\right)\}} \Big\{(x_1,\dots,x_k,x_{k+1}) \mid \mathbf{x} \in S,\ f^{(k)}(\mathbf{x}) = m' \text{ et } x_{k+1} \in v(a)\Big\} \\ &= \bigcup_{\{a\in A \mid a=\left((k,m'),(k+1,m)\right)\}} S_{(k,m')} \times \{v(a)\}. \end{aligned} \tag{2.3}$$

Supposons que l'égalité (2.2) soit vérifiée pour $k \leq K$, alors comme les unions sont disjointes on obtient :

$$\begin{aligned} |S_{(k+1,m)}| &= \sum_{\left\{a\in A \mid a=\left((k,m'),(k+1,m)\right)\right\}} |S_{(k,m')}| \times |v(a)| \\ &= \sum_{\left\{a\in A \mid a=\left((k,m'),(k+1,m)\right)\right\}} |Card((k,m'))| \times |v(a)| \\ &= Card((k+1,m)). \end{aligned} \tag{2.4}$$

Pour conclure, il suffit de remarquer que $S_{(K,M)} = \psi(D)$ ce qui implique $Card((K,M)) = |\psi(D)|$. □

Nous aurions tout aussi bien pu définir l'algorithme `CardStates` "de droite à gauche", c'est-à-dire en commençant par affecter 1 au nœud destination $(K,M)$, c'est alors dans le nœud source $(0,0_{\mathcal{G}})$ qu'on aurait retrouvé la valeur $|\psi(D)|$.

### Transformation d'un diagramme en un ensemble d'états

L'algorithme `DiagramToStates` prend en entrée un diagramme $D$ et retourne l'ensemble $\psi(D)$. Il est similaire à l'algorithme 5 présenté précédemment : l'idée est comme dans `CardStates` d'attribuer des valeurs aux nœuds itérativement en fonction de ses arcs entrants. Ici on leur attribue non plus un nombre mais un ensemble d'états que l'on note $v_{\mathcal{N}}(n)$. L'algorithme retourne $v_{\mathcal{N}}((K,M))$, qui correspond à l'ensemble $\psi(D)$.

**Algorithme 6:** `DiagramToStates`

**Données :** $D = (\mathcal{N}, A) \subseteq \mathcal{D}(K,M)$, $v$
**Résultat :** $\psi(D) \subseteq \Omega(K,M)$

```
début
    pour chaque n ∈ N faire
        v_N(n) ← ∅;
    pour chaque a = ((0,ℓ),(1,m)) ∈ A(1) faire
        v_N((1,m)) ← v(a) ;
    pour k = 2,...,K faire
        pour chaque a = ((k−1,ℓ),(k,m)) ∈ A(k) faire
            v_N((k,m)) ← v_N((k,m)) ∪ v_N((k−1,ℓ)) × v(a);
    renvoyer v_N((K,M)).
```

Dans le nœud destination, il y a $\Omega(K,M)$ d'opérations de produit à faire. Dans chaque colonne $k < K$, on construit les vecteurs $\mathbf{x} \in \Omega(k,m)$ par une opération produit. Or

$|\Omega(k,m)| < |\Omega(K,M)|$, il y a donc au plus $\Omega(K,M)$ d'opérations de produit à effectuer pour chaque colonne $k$. La complexité en temps de calcul de cet algorithme est donc en $O(K|\Omega(K,M)|)$. En procédant comme dans la démonstration du lemme 6, on prouve que `DiagramToStates` renvoie bien $\psi(D)$ en posant $S_{(k,m)} := v_{\mathcal{N}}((k,m))$.

### Marche aléatoire dans les diagrammes

Supposons que pour chaque $k \in \{1,\dots,K\}$ on dispose d'une fonction $w_k : \mathcal{E}_k \to \mathbb{R}^+$ qui attribue un poids à chaque $x_k \in \mathcal{E}_k$ et d'une fonction $w : \Omega(K,M) \to \mathbb{R}^+$ telle que

$$w(\mathbf{x}) = \prod_{k=1}^{K} w_k(x_k).$$

L'algorithme `RandState` (algorithme 9) est un échantillonneur de $\psi(D)$ selon une distribution à forme produit. Il produit un état $\mathbf{x} \in \psi(D)$ avec probabilité

$$p_{\mathbf{x}} = \frac{1}{W} \prod_{k=1}^{K} w_k(x_k) \quad \text{avec} \quad W = \sum_{\mathbf{x} \in \psi(D)} \prod_{k=1}^{K} w_k(x_k). \tag{2.5}$$

Pour ce faire, `RandState` effectue une marche aléatoire dans le diagramme. Avant d'effectuer cette marche on doit avoir préalablement calculé un poids pour chaque arc et un poids pour chaque nœud du diagramme. C'est ce que font respectivement les algorithmes `WeightArcs` et `WeightNodes`. Commençons par décrire ces deux algorithmes.

**Calcul des poids des arcs** Chaque arc $a \in A(k)$ du diagramme présent en colonne $k$ représente un ensemble de valeur $v(a) \subseteq \mathcal{E}_k$. L'algorithme `WeightArcs` attribue à chaque arc du diagramme un poids correspondant à la somme des poids des éléments de l'ensemble $v(a)$. En supposant que pour chaque arc du diagramme, $v(a)$ s'obtienne avec une complexité en $O(1)$ alors `WeightArcs` a une complexité en temps de calcul en $O(\sum_{a\in A} |v(a)|)$.

**Algorithme 7:** `WeightArcs`

**Données :** $D = (\mathcal{N}, A) \subseteq \mathcal{D}(K,M)$, $v$, $w_1, \dots, w_K$

1 **début**
2  **pour *chaque*** $a := \big((k-1,m),(k,m')\big) \in A$ **faire**
3  $w_A(a) \leftarrow \sum_{e \in v(a)} w_k(e)$;
4  **renvoyer** $w_A$.

**Calcul des poids des nœuds** L'algorithme `WeightNodes` attribue à chaque nœud du diagramme un poids en fonction de ses arcs entrants. Il procède colonne par colonne en commençant par la dernière colonne. Il commence par attribuer la valeur 1 au nœud destination puis calcule la valeur de chaque nœuds en fonction de ses nœuds voisins dans la colonne suivante et de la valeur des arcs le liant à ses voisins (voir ligne 7).

En supposant que pour chaque arc $a \in A$ du diagramme, $w_A(a)$ s'obtienne avec une complexité en $O(1)$ alors `WeightNodes` a une complexité en temps de calcul en $O(|A|)$.

**Lemme 7.** *Soient $D = (\mathcal{N}, A) \subseteq \mathcal{D}(K,M)$ un diagramme et $w_A : A \to \mathbb{R}^+$ une fonction telle que pour tout $a \in A$,*

$$w_A(a) = \sum_{e \in v(a)} w_k(e).$$

**Algorithme 8:** `WeightNodes`

**Données** : $D = (\mathcal{N}, A) \subseteq \mathcal{D}(K, M)$, $w_A$

**début**
  **pour** ***chaque*** $n \in \mathcal{N}$ **faire**
    $w_{\mathcal{N}}(n) \leftarrow 0$;
  $w_{\mathcal{N}}((K, M)) \leftarrow 1$;
  **pour** $k = K, K-1, \ldots, 1$ **faire**
    **pour** ***chaque*** $a = \big((k-1, m), (k, m')\big) \in A(k)$ **faire**
      $w_{\mathcal{N}}((k-1, m)) \leftarrow w_{\mathcal{N}}((k-1, m)) + w_A(a) w_{\mathcal{N}}((k, m'))$;
  **renvoyer** $w_{\mathcal{N}}$.

*Alors l'algorithme* `WeightNodes` *affecte au nœud source la valeur $W$ définie par l'équation* (2.5).

*Démonstration.* Pour prouver le lemme 7, on procède de la même manière que dans la preuve du lemme 6. On commence par poser : $\overline{S}_{(K,M)} = \{\epsilon\}$ et pour tout $k$ tel que $0 \leq k < K$ :

$$\overline{S}_{(k,m)} := \Big\{ (x_k, x_{k+1}, \ldots, x_K) \mid f^{(k)}(\mathbf{x}) = m \text{ et } \mathbf{x} \in \psi(D) \Big\}.$$

On montre ensuite par récurrence pour $k = K, K-1 \ldots 1, 0$ que pour tout nœud $(k, m)$ du diagramme on a

$$w_{\mathcal{N}}((k, m)) = \sum_{\mathbf{x} \in \overline{S}_{(k,m)}} \prod_{i=k}^{K} w_i(x_i).$$

Ceci ce fait en remarquant que

$$\overline{S}_{(k,m)} = \bigcup_{\{a \in A \mid a = \big((k,m'),(k+1,m)\big)\}} \overline{S}_{(k+1,m')} \times \{v(a)\}.$$

Enfin comme $\overline{S}_{(0,0_{\mathcal{G}})} = \psi(D)$ on en déduit que $w_{\mathcal{N}}((0, 0_{\mathcal{G}})) = W$. □

**Algorithme** `RandState` Soit un diagramme $D = (\mathcal{N}, A)$, supposons que l'on dispose des fonctions $w_A$ et $w_{\mathcal{N}}$ calculées à l'aide des algorithmes `WeightArcs` et `WeightNodes`. L'algorithme `RandState` produit un état $\mathbf{x} \in \psi(D)$ selon la distribution à forme produit décrite par (2.5) en effectuant une marche aléatoire dans le diagramme. Cette marche commence au nœud source et se termine au nœud destination. Chaque arc $\big((k-1, m), (k, m')\big)$ est choisi avec une probabilité $\frac{w_A(a) w_{\mathcal{N}}(n_k)}{w_{\mathcal{N}}(n_{k-1})}$. Pour chaque arc $a \in A(k)$ emprunté, un $x_k$ est choisi aléatoirement dans l'ensemble $v(a)$ avec probabilité $\frac{w_k(e)}{w_A(a)}$. On construit ainsi un état $\mathbf{x} = (x_1, \ldots, x_K)$ en parcourant le diagramme.

Supposons que le temps d'accès aux poids des arcs et des nœuds soit en $O(1)$ et que les deux tirages aléatoires (arc et élément) se fassent avec une complexité en $O(T)$ alors (une fois calculé $w_A$ et $w_{\mathcal{N}}$), cet algorithme a une complexité en $O(KT)$.

**Lemme 8.** *Soit $D \subseteq \mathcal{D}(K, M)$ un diagramme et une fonction $w_A : A \to \mathbb{R}$ telle que pour tout $a \in A(k)$, $w_A(a) = \sum_{e \in v(a)} w_k(e)$. Alors l'algorithme* `RandState` *retourne l'état $\mathbf{x} \in \psi(D)$ avec probabilité $p_{\mathbf{x}}$ telle que*

$$p_{\mathbf{x}} = \frac{1}{W} \prod_{k=1}^{K} w_k(x_k) \quad \textit{avec} \quad W = \sum_{\mathbf{x} \in \psi(D)} \prod_{k=1}^{K} w_k(x_k).$$

**Algorithme 9:** `RandState`

**Données** : $D = (\mathcal{N}, A) \subseteq \mathcal{D}(K, M)$, $w_A$, $w_{\mathcal{N}}$
**Résultat** : $\mathbf{x} \in \Omega(K, M)$

**début**
  $n_0 \leftarrow (0, 0_{\mathcal{G}})$;
  $\mathbf{x} \leftarrow ()$ ;
  **pour** $k = 1, 2, \ldots, K$ **faire**
    Choisir un arc $a = (n_{k-1}, n_k) \in A(k)$ avec probabilité $\frac{w_A(a) w_{\mathcal{N}}(n_k)}{w_{\mathcal{N}}(n_{k-1})}$;
    Choisir un élément $e \in v(a)$ avec probabilité $\frac{w_k(e)}{w_A(a)}$ ;
    $\mathbf{x} \leftarrow \mathbf{x}.e$;
  **renvoyer** $\mathbf{x}$.

*Démonstration.* Considérons $\mathbf{x} \in \psi(D)$. Il existe donc un unique chemin associé à $\mathbf{x}$ dans $D$ partant du nœud source et arrivant au nœud destination. Numérotons les nœuds puis les arcs de la façon suivante : $n_0 = (0, 0_{\mathcal{G}})$, $n_K = (K, M)$ et

$$g(\mathbf{x}) = \left\{ (n_0, n_1), (n_1, n_2), \ldots, (n_{K-1}, n_K) \right\} = \{a_1, a_2, \ldots, a_K\}.$$

La probabilité pour l'algorithme 9 de retourner $\mathbf{x}$ est donnée par :

$$\begin{aligned} p_{\mathbf{x}} =& \frac{w_A(a_1) w_{\mathcal{N}}(n_1) w_1(x_1)}{w_{\mathcal{N}}(n_0) w_A(a_1)} \times \cdots \times \frac{w_A(a_K) w_{\mathcal{N}}(n_K) w_K(x_K)}{w_{\mathcal{N}}(n_{K-1}) w_A(a_K)} \\ =& \frac{w_{\mathcal{N}}(n_K)}{w_{\mathcal{N}}(n_0)} \prod_{k=1}^{K} w_k(x_k) \\ =& \frac{w_{\mathcal{N}}((K, M))}{w_{\mathcal{N}}((0, 0_{\mathcal{G}}))} \prod_{k=1}^{K} w_k(x_k) \\ =& \frac{1}{W} \prod_{k=1}^{K} w_k(x_k). \end{aligned} \tag{2.6}$$

La dernière ligne se déduit en appliquant le lemme 7. □

**Génération uniforme dans $\psi(D)$** Pour générer uniformément un état $\mathbf{x} \in \psi(D)$, il suffit de choisir pour tout $k$, $w_k(x_k) = 1$. En effet, dans ce cas l'algorithme `WeightNode` correspond donc à l'algorithme `CardStates` en traitant les colonnes du diagramme de droite à gauche (avec un $k$ décroissant). On a alors $w_{\mathcal{N}}((0, 0_{\mathcal{G}})) = |\psi(D)|$. La probabilité pour chaque état $\mathbf{x} \in \psi(D)$ d'être obtenu par `RandState` est $p_{\mathbf{x}} = (w_{\mathcal{N}}((0, 0_{\mathcal{G}})))^{-1} = |\psi(D)|^{-1}$. Ainsi, `RandState` retourne un état uniformément distribué dans l'ensemble $\psi(D)$.

### Exemples

On présente maintenant des exemples d'utilisation de diagrammes. Ces derniers mettent en évidence le lien entre programmation dynamique et la structure de donnée diagramme.

Dans notre premier exemple, on montre que l'on peut échantillonner la distribution stationnaire d'un réseau de Gordon et Newell en utilisant l'algorithme `RandState`. On calculera notamment la constante de normalisation à l'aide de l'algorithme `WeigthNodes`. On verra pour ce problème que la complexité en temps de calcul de la constante de normalisation par `WeigthNodes` correspond à celle de l'algorithme de Buzen [19]. On s'intéressera ensuite à un autre célèbre algorithme de programmation dynamique, l'algorithme de Held et Karp

[36]. Ce dernier permet de résoudre le problème du voyageur de commerce et sa complexité mémoire est étonnamment liée à celle d'un diagramme complet des permutations. Enfin on reviendra sur l'exemple de la section 2.1.3 et on s'intéressera à la simulation d'une chaîne de Markov à horizon fixé, on utilisera une variante de l'algorithme `RandState`.

**Réseaux de Gordon et Newell** Dans le chapitre 1 nous nous sommes intéressés aux réseaux fermés de Gordon et Newell [33, 38]. On a vu qu'un état d'un tel réseau à $K$ files et $M$ clients est modélisé par un vecteur $\mathbf{x} \in \mathbb{N}^K$ dans lequel chaque composante $\mathbf{x}_k$ représente le nombre de clients en file $k$ et $\sum_{k=1}^{K} x_k = M$. Dans le paragraphe 2.1.2 nous nous sommes intéressés à la suite sans mémoire des sommes cumulées et avons vu que :

$$\Omega(K,M)_{cum} = \left\{ \mathbf{x} = (x_1, x_2, \ldots, x_K) \in \mathbb{N}^K \mid \sum_{k=1}^{K} x_k = M \right\} = \mathcal{S}.$$

L'espace des états d'un réseau de Gordon et Newell à $K$ files et $M$ clients correspond donc à l'espace sans mémoire ($\mathcal{S} = \Omega(K,M)_{cum}$).

Considérons $D = \mathcal{D}(K,M)$ le digramme complet des sommes cumulées. On a alors pour tout arc $a \in A$, $|v(a)| = 1$ et $|A| = \frac{(K-2)(M+2)(M+1)}{2} + 2(M+1)$.

Considérons un réseau de Gordon et Newell à $K$ files et $M$ clients. Le nombre d'états de ce réseau peut être calculé par `CardStates`($D$) avec une complexité en $O(KM^2)$.

Toujours dans le chapitre 1, nous avons vu de plus que la probabilité stationnaire d'un réseau de Gordon et Newell [33, 38] est à forme produit. Dans sa forme générale, elle est donnée par :

$$\pi_{\mathbf{x}} = \frac{1}{G(K,M)} \prod_{k=1}^{K} f_k(x_k) \quad \text{avec} \quad G(K,M) = \sum_{\mathbf{x} \in \mathcal{S}} \prod_{k=1}^{K} f_k(x_k).$$

Pour chaque file $k$ du réseau, la fonction $f_k : \mathbb{N} \to \mathbb{R}^+$ attribue un poids à la composante $x_k$ d'un vecteur $\mathbf{x}$. Elle ne dépend que de paramètres internes à la file : son taux de service et son nombre de serveurs.

En posant $w_k := f_k$ dans l'équation (2.5), comme $\mathcal{S} = \psi(D)$ on en déduit que $W = G(K,M)$. Ainsi l'algorithme `WeightNodes` calcule $G(K,M)$ avec une complexité en $O(KM^2)$. Dans [19], Buzen calcule la constante de normalisation dans le cas multi-serveurs en effectuant $2KM(M+1)$ opérations ce qui correspond à la complexité en temps de calcul de `WeightNodes`.

Après avoir calculé $w_A$ et $w_{\mathcal{N}}$ à l'aide respectivement de `WeightArcs` et de `WeightNodes`, on peut alors utiliser l'algorithme `RandState` pour échantillonner la distribution stationnaire du réseau de Gordon Newell.

**Algorithme de Held et Karp** En 1962, Held et Karp [36] ont montré que l'on peut résoudre le problème du voyageur de commerce pour $n$ villes à l'aide de la programmation dynamique en utilisant $(n-1)2^{n-2}$ allocations mémoires. Étonnamment, cette complexité correspond exactement à celle d'un diagramme complet des permutations pour $K = n-1$. Essayons de comprendre pourquoi.

Soit un ensemble de $n$ villes et une ville de départ, on cherche à déterminer le circuit le plus court qui visite chaque ville une et une seule fois et se qui termine dans la ville de départ. C'est le problème du voyageur de commerce. Numérotons les villes de $1$ à $n$. Notons

$d(i,j)$ la distance entre la ville $i$ et la ville $j$. Le but est donc de trouver la permutation $\sigma$ de l'ensemble $\{1,2,\ldots,K\}$ qui commence par $1$ et minimise :

$$d(\sigma) := d(1,\sigma(2)) + \sum_{i=2}^{n-1} d(\sigma(i),\sigma(i+1)) + d(\sigma(n),1).$$

Dans le problème de Held et Karp, la première ville est fixée. On s'intéresse donc aux permutations de taille $n-1$. Illustrons graphiquement l'algorithme de résolution de Held et Karp sur le diagramme complet des permutations tel que $K = n-1$, $\mathcal{M} := \{2,\ldots,K\}$ et $0_{\mathcal{G}} = \{1\}$. On commence par attribuer à chaque nœud d'un couple de valeurs. Ce couple de valeurs enregistre la dernière ville visitée ainsi que la plus petite distance requise pour parcourir les villes de l'ensemble contenu dans le nœud. La ville $1$ débute le circuit et $\mathcal{M} := \{1,2,\ldots,K\}$ est l'ensemble des villes à parcourir. Considérons $\mathcal{D} = (K,\mathcal{M})$ le diagramme complet des permutations. Pour chaque nœud $(k,m)$ du diagramme, on associe un couple ville et distance $(v_{k,m}, D_{k,m})$ calculé récursivement : $v_{k,m}$ est la dernière ville visitée dans le parcours contenant les villes de l'ensemble $m$ et $D_{k,m}$ est la distance parcourue pour visiter toutes les villes de l'ensemble $m$. Pour simplifier la notation, on note $i = v_{k-1,m'}$ et $j = m \setminus m'$ et on procède de la manière suivante :

$$D_{k,m} = \begin{cases} 0 & \text{si } k = 0 \\ \min\{d(i,j) + D_{k-1,m'} + d_{j,0} \mid \big((k-1,m'),(k,m)\big) \in A\} & \text{si } k = K-1 \\ \min\{d(i,j) + D_{k-1,m'} \mid \big((k-1,m'),(k,m)\big) \in A\} & \text{sinon,} \end{cases}$$

et

$$v_{k,m} = j \text{ minimisant } D_{k,m}.$$

La figure ci-dessous illustre l'application de l'algorithme de Held et Karp sur un diagramme pour un exemple contenant 4 villes. La carte des villes est donnée dans le graphe de gauche. Les quatre diagrammes illustrent le calcul de la valeur des nœuds pour $k = 1,2,3,4$. On en déduit que les deux circuits minimisant la distance parcourue sont de poids 14. Le premier consiste à effectuer le trajet suivant : Bravos, Pentos, Norvos, Lorath, Bravos et le second le trajet inverse : Bravos, Lorath, Norvos, Pentos, Bravos.

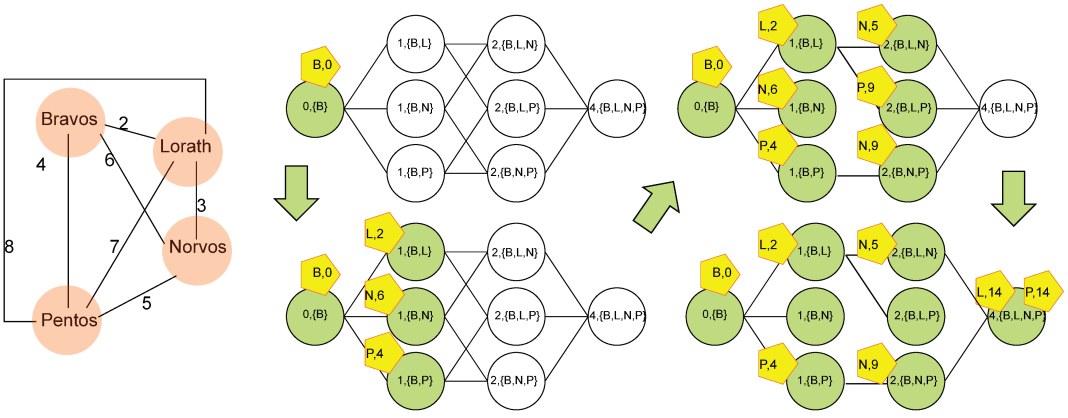



**FIGURE 2.6:** Algorithme de Held et Karp illustré par un diagramme complet des permutations.

L'algorithme de Held et Karp se réduit donc en un algorithme de recherche du plus court chemin entre le nœud source et le nœuds destination dans le diagrammes complet des permutations.

**Simulation d'une chaîne de Markov à horizon fixé** Reprenons l'exemple 7 de la simulation de file d'attente $M/M/2/10$. Pour $M = 0$ et $K = 5$ le diagramme complet est celui illustré

en figure 2.7. On veut maintenant simuler la chaîne de Markov à horizon (et temps) fixé(s) pour une simulation de durée 5 et pour un départ fixé à $X_0 = 0$ et un horizon fixé à $X_5 = 0$. C'est-à-dire, on veut être en mesure de produire (avec les bonnes probabilités) des vecteurs $\mathbf{u} = (u_1, u_2, u_3, u_4, u_5)$ tels que $X_0 = 0$, $X_1 := \mathrm{maj}(X_0, u_1)$,..., $X_5 := \mathrm{maj}(X_4, u_5) = 0$. Pour ce faire on se propose d'utiliser l'algorithme `RandState`.

La valuation d'un arc $a = \big((k-1, s), (k, s')\big)$ a été définie pour $\mathcal{E}_k$ discret par :

$$v(a) = \{u_k \in \mathcal{E}_k \mid f_{mar}^{(k)}(u_1, \ldots, u_{k-1}) = s \text{ et } f_{mar}^{(k+1)}(u_1, \ldots, u_{k-1}, u_k) = s'\} \subseteq \mathcal{E}_k.$$

Dans le ce cas présent, $\mathcal{E}_k := [0, 1[$ n'est pas discret mais on conserve la même définition.

On modifie la ligne 3 de l'algorithme `WeightArcs` afin que le poids de chaque arc $a \in A$ soit égal à la mesure de $v(a)$. On pose alors :
(ligne 3) $w_A(a) := \int_0^1 \mathbb{1}_{\{y \in v(a)\}}(y)\, \mathrm{d}y$.

On modifie ensuite la ligne 6 de l'algorithme `RandState` afin de choisir uniformément $x_k$ dans $v(a)$. Ceci donne
(ligne 6) `Choisir` $e \in v(a)$ `avec probabilité` $\frac{1}{w_A(a)}$ ;

Dans la figure 2.7, la mesure de l'arc $\big((0,0),(1,1)\big)$ est égale à $1 - 2a$. Le nouvel algorithme `RandState` appliqué au diagramme complet de la figure 2.7 produit donc un vecteur $\mathbf{u} = (u_1, u_2, u_3, u_4, u_5)$. Ce vecteur peut être utilisé pour simuler une trajectoire de la chaîne de Markov $(X_n)_{n \leq 5}$ modélisant la file $M/M/2/10$ et pour laquelle on a $X_0 = 0$ et $X_5 = 0$.

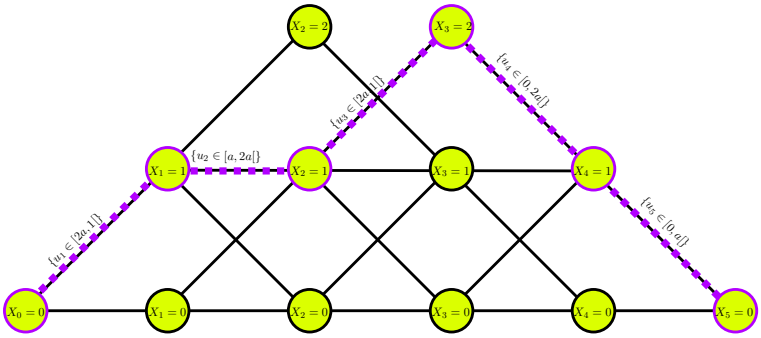



**FIGURE 2.7:** Digramme complet.

# Conclusion

La représentation par diagrammes permet la factorisation des suffixes et des préfixes des états. Ainsi, bien souvent la complexité de la représentation des diagrammes est beaucoup plus faible que celle des ensembles d'états. Par exemple on peut représenter les $\binom{K+M-1}{K-1}$ états d'un réseaux fermés de files d'attente avec $O(KM^2)$ arcs du diagramme complet des sommes cumulées. Le prix à payer en utilisant la représentation par diagramme (quand le diagramme n'est pas complet) est une possible sur-représentation de l'espace des états.

Dans un diagramme, la valuation de chaque arc dépend des nœuds de départ et d'arrivée de ce dernier. Par exemple, dans un diagramme des maxima, chaque arc $a = \big((k-1, m), (k, m')\big)$ a pour valeur $v(a) = \{m, m+1, \ldots, m'\}$ et encode toujours $m' - m + 1$ valeurs de $x_k$. Afin de pouvoir affiner le nombre de valeurs encodées par chaque arc d'un diagramme, on propose

une nouvelle définition des diagrammes qui intègre directement la fonction de valuation à la structure diagramme.

**Définition 6.** *On appelle* ***diagramme valué*** *un diagramme $D = (\mathcal{N}, A)$ auquel est explicitement associé une fonction de valuation $p$ telle que pour tout $a \in A$ on ait*

$$p(a) \neq \emptyset \quad \textit{et} \quad p(a) \subseteq v(a).$$

*Un tel diagramme est noté $D = (\mathcal{N}, A, p)$.*

Les diagrammes valués pourront faire l'objet d'un travail encore plus général concernant les diagrammes. Ce travail pourrait de plus intégrer le cas des valuations non discrètes ce qui permettrait d'approfondir le sujet de la simulation d'une chaîne de Markov à horizon fixé.

Les algorithmes présentés ci-dessus utilisent le paradigme de programmation dynamique, ils permettent le calcul du nombre d'états et l'échantillonnage des états représentés par un diagramme. On a notamment constaté le lien entre les diagramme et deux célèbres algorithmes de programmation dynamique qui sont les algorithme de Held et Karp [36] et celui de Buzen [19]. Les digrammes seront utilisés dans les trois chapitres suivants dans le cadre de la simulation parfaite des réseaux fermés de files d'attentes et joueront le rôle d'enveloppe. Grâce à la représentation par diagramme, on représentera les ensembles d'états et on effectuera les mise à jour à moindre coût.

Nous venons d'introduire un cadre général pour l'utilisation des diagrammes. À partir de maintenant, les diagrammes auront une suite sans mémoire $\mathcal{F}$ qui sera explicitement donnée et chaque arc aura une valuation de cardinalité 1. La suite des sommes cumulées $\mathcal{F}_{cum}$ sera utilisé au chapitre 3. Le diagramme issu de cette suite sans mémoire sera utilisé comme enveloppe pour la simulation parfaite des réseaux fermés de files d'attente monoclasses. Le chapitre 4 utilisera la suite des sommes vectorielles cumulées $\mathcal{F}_{vec}$ dans le but de simulation parfaite dans le cas multiclasse. Dans le chapitre 7 on définira une suite sans mémoire non exposée dans ce chapitre. Elle aura pour objet les partitions d'entiers. Ses diagrammes associés seront utilisés pour l'échantillonnage de Boltzmann des multi-ensembles de cardinalité fixe. Dans le chapitre 5 on définira le *diagramme aggloméré* qui consiste en la concaténation de plusieurs diagrammes. On verra que certains algorithmes présentés dans ce chapitre pourront êtres adaptés aux diagrammes agglomérés. Enfin dans le chapitre 8 on présentera le *package* Python `DiagramS` qui a été conçu suite à la rédaction de ce chapitre et dans la philosophie de ce dernier. Dans `DiagramS` un diagramme est traité de manière générique, les algorithmes que nous venons de présenter sont implémentés qu'une seule fois et agissent sur tout type de diagrammes.

# 3 Simulation parfaite de réseaux fermés monoclasses

Le chapitre 3 est consacré à la présentation d'une nouvelle technique de simulation parfaite pour des réseaux fermés de files d'attente monoclasses, multi-serveurs et à capacités finies.

La première section est une introduction au chapitre. On commencera par décrire le réseau étudié et la chaîne de Markov qui modélise la dynamique du réseau. L'algorithme de Propp et Wilson pour cette chaîne sera ensuite présenté et nommé algorithme PSS. Comme on a vu au chapitre 1, la taille de l'espace des états de la chaîne de Markov rend l'algorithme de Propp et Wilson irréalisable en pratique. La deuxième section constitue le cœur du chapitre, elle est consacrée à l'algorithme PSD qui est un algorithme de simulation parfaite tirant avantage de la représentation par diagrammes et permettant ainsi la réalisation de simulation parfaite pour notre modèle. La stratégie développée consiste à encoder les ensembles d'états par des diagrammes et à réaliser la simulation directement sur le diagramme. On présentera à cet effet un algorithme de transition qui effectue les transitions directement sur les diagrammes avec une complexité en $O(KM^2)$. La troisième section est consacrée à une représentation encore plus compacte de l'espace des états que l'on appelle *diagramme sans trou*. Une transition sur un diagramme sans trou s'effectue avec une complexité en $O(KM)$. On présentera PSF, l'algorithme de simulation parfaite utilisant les diagrammes sans trou. En section quatre, on comparera numériquement les performances algorithmiques en s'intéressant aux moyennes du nombre de transitions utilisé par les algorithmes PSS, PSD et PSF. On mettra en œuvre des heuristiques afin de réduire ce nombre pour les algorithmes ayant recours aux diagrammes. Enfin, afin de justifier que les algorithmes PSD et PSF se terminent, on démontrera pour le cas des diagramme sans trou et avec trou, qu'il existe une suite couplante de transitions qui transforment le diagramme complet en un diagramme représentant un seul état. Ceci fera l'objet la dernière section.

## 3.1 Introduction

### 3.1.1 Description du réseau

On considère un réseau fermé de $K$ files d'attente $./M/1/C$ possédant $M$ clients au total. Chaque file $k \in \mathcal{Q} := \{1, \ldots, K\}$ a une capacité finie $C_k$ et un nombre de serveurs $E_k$. Le temps de service dans chaque file est exponentiel et indépendant de l'état du réseau ou du temps de service des autres serveurs. Pour simplifier, on fait l'hypothèse que chaque serveur dans une même file dispose d'un même taux de service $\mu_k$. Après qu'un client a été servi en file $i$, il demande à être dirigé en file $j$ avec probabilité $p_{i,j}$ indépendamment encore de l'état du réseau et de l'évolution passée du réseau. Si la file $j$ est pleine alors le client reste en file $i$ et perd son service. Pour repartir le client devra attendre d'être resservi. Une nouvelle destination sera alors choisie indépendamment de $j$. Cette discipline de service est connue sous le nom de RS-RD (*repetitive service - random destination*). La matrice de routage $\mathbf{P} = (p_{i,j})_{i,j \in \mathcal{Q}}$ des clients dans le réseau est stochastique. Nous nous intéressons seulement aux régimes stationnaires de tels réseaux aussi nous considérons que $\mathbf{P}$ est irréductible. Notons $R = \{(i,j) \text{ tel que } p_{i,j} > 0\}$, le réseau peut alors être représenté par un graphe orienté $G = (\mathcal{Q}, R)$. Ce dernier est fortement connexe car $\mathbf{P}$ est irréductible. Un exemple de réseau est donné en figure 3.1. Un état du réseau est représenté par un

vecteur $x \in \mathbb{N}^K$ dans lequel la composante $x_k$ est le nombre de clients en file $k$. L'espace des états du réseau est donné par

$$\mathcal{S} = \left\{ \mathbf{x} = (x_1, x_2, \ldots, x_K) \in \mathbb{N}^K \mid \sum_{k=1}^{K} x_k = M \text{ et } \forall k,\ 0 \leq x_k \leq C_k \right\}.$$

Si toutes les files sont de capacité infinie, c'est-à-dire si pour chaque $k \in \mathcal{Q}$ on a $C_k = M$ alors $|\mathcal{S}| = \binom{K+M-1}{K-1}$. Ainsi nous en déduisons pour le cas général $|\mathcal{S}| \leq \binom{K+M-1}{K-1}$, la cardinalité de $\mathcal{S}$ est donc en $O(M^K)$ pour un réseau contenant un nombre négligeable de files en comparaison du nombre de clients ($K \ll M$).

**Exemple 10.** *Considérons le réseau de la figure 3.1, ce dernier possède $K = 5$ files. Pour $M = 3$ et $\mathbf{C} = (2, 1, 3, 1, 2)$ (vecteur des capacités), le nombre d'états possibles du réseau est $|\mathcal{S}| = 23$. Par exemple $\mathbf{x} = (0, 0, 2, 0, 1)$ est un état du réseau.*

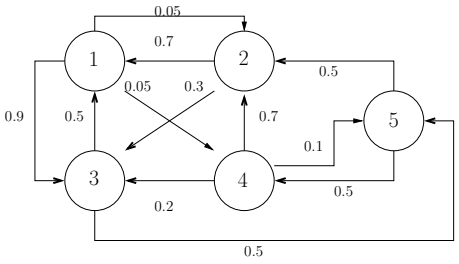



**FIGURE 3.1:** Réseau fermé avec 5 files.

Si le nombre de clients dans la file $k$ est $m \geq E_k$, alors $E_k$ clients sont en service et le taux de service global dans la file $k$ est égal à $E_k \mu_k$. Dans le cas contraire, si $m < E_k$ alors le taux de service global est égal à $m\mu_k$. Pour un couple de files $(i, j) \in \mathcal{Q}^2$ et $s \in \{1, \ldots, E_k\}$, la fonction $t_{i,j,s}$ modélise le routage d'un client de la file $i$ vers la file $j$ quand le nombre de clients en file $i$ est au moins égal à $s$.

$$t_{i,j,s}(\mathbf{x}) = \mathbf{x} + (\mathbf{e_j} - \mathbf{e_i}) \mathbb{1}_{\{x_i \geq s \text{ et } x_j < C_j\}}.$$

La fonction $t_{i,j,1}$ modélise le routage d'un seul client, on note alors $t_{i,j} := t_{i,j,1}$. On définit respectivement les transitions associées aux triplets $(i, j, s)$ et $(i, j, 1)$ (pour $s = 1$) sur un ensemble d'état $S \subseteq \mathcal{S}$ de la manière suivante

$$t_{i,j,s}(S) = \bigcup_{\mathbf{x} \in \mathcal{S}} t_{i,j,s}(\mathbf{x}) \qquad \text{et} \qquad t_{i,j}(S) = \bigcup_{\mathbf{x} \in \mathcal{S}} t_{i,j}(\mathbf{x}).$$

### 3.1.2 Chaîne de Markov

Après uniformisation de la chaîne de Markov a temps continu par le taux maximum de départ, la dynamique du système est modélisée par la chaîne de Markov $(X_n)_{n \in \mathbb{N}}$ à valeur dans $\mathcal{S}$ avec

$$\begin{cases} X_0 \in \mathcal{S}, \\ X_{n+1} = t_{i,j,s}(X_n) \text{ avec probabilité } \frac{\mu_i p_{i,j}}{\sum_{k \in \mathcal{Q}} E_k \mu_k}. \end{cases}$$

Chaque triplet $(i, j, s)$ est choisi avec probabilité $\frac{\mu_i p_{i,j}}{\sum_{k \in Q} E_k \mu_k}$ et la transition $t_{i,j,s}$ est réalisée sur l'état courant.

Comme la matrice des transitions $\mathbf{P}$ est irréductible, la chaîne $(X_n)_{n \in \mathbb{N}}$ est ergodique. Notre objectif est maintenant d'échantillonner la distribution stationnaire de $(X_n)_{n \in \mathbb{N}}$ en utilisant l'algorithme de simulation parfaite.

### 3.1.3 Première approche pour la simulation parfaite

On note $(U_{-n})_{n \in \mathbb{N}}$ une suite de variable aléatoire i.i.d. telle que $\mathbb{P}(U_0 = (i, j, s)) = \frac{\mu_i p_{i,j}}{\sum_{k \in Q} E_k \mu_k}$.

**Algorithme 10:** Simulation parfaite sur l'espace des états (PSS)

**Données :** $(U_{-n} = (i_{-n}, j_{-n}, s_{-n}))_{n \in \mathbb{N}}$ une suite i.i.d de v.a.
**Résultat :** $\mathbf{x} \in \mathcal{S}$

```
début
    n ← 1;
    t ← t_{U_{-1}};
    S ← t(S) ;
    tant que |S| ≠ 1 faire
        n ← 2n;
        t ← t_{U_{-1}} ∘ ··· ∘ t_{U_{-n}};
        S ← t(S) ;
    renvoyer x, l'unique élément de S.
```

**Théorème 5** (Application directe de [51])**.** *L'algorithme 10 (PSS) se termine avec probabilité* 1 *en un temps d'espérance finie et retourne* $\mathbf{x} \in \mathcal{S}$ *distribué selon la distribution stationnaire de* $(X_n)_{n \in \mathbb{N}}$.

L'algorithme 10 est celui décrit par Propp et Wilson adapté à au modèle que l'on vient de présenter. Dans cette version, on n'utilise pas de tableau et on double le temps de simulation ($(N_k)_k = \{1, 2, 4, \ldots\}$). Comme on l'a vu au chapitre 1, la complexité en temps de calcul de l'algorithme de Propp et Wilson est liée au cardinal de l'espace des états. Dans le cas présent $|\mathcal{S}|$ a une complexité en $O(M^K)$, ce qui rend cet algorithme inutilisable en pratique. On a vu dans le chapitre 1 que la contrainte globale sur le nombre total de clients dans le réseau ($\sum_{k=1}^{K} x_k = M$) ne permet pas l'application de techniques telles que les enveloppes. Cependant, on verra dans la section suivante que cette même contrainte peut être exploitée et rendre de ce fait la simulation parfaite possible.

## 3.2 Simulation parfaite à l'aide des diagrammes

Cette section présente une nouvelle technique pour simulation parfaite des réseaux fermés de files d'attente. Elle se base sur la représentation par diagramme qui a une complexité en $O(KM^2)$.

Montrons maintenant que l'espace des états $\mathcal{S}$ peut être défini à partir d'une suite sans mémoire.

### 3.2.1 Espace des états et diagramme

Considérons $\mathcal{E} = \mathcal{G} = \mathbb{N}$ et $\forall k,\ k \leq K,\ \mathcal{E}_k = \{1, \dots, C_k\}$ et

$$\begin{array}{rccc} f^{(k)} & : & \mathbb{N}^k & \to & \mathbb{N} \\ & & \mathbf{x} & \mapsto & \sum_{i=0}^{k} x_i. \end{array}$$

La suite $\mathcal{F} := (f^{(k)})_{k\in\mathbb{N}}$ est sans mémoire car elle s'inscrit dans le cadre du lemme 2 si on considère le monoïde $(\mathbb{N}, +, 0)$. De plus, l'espace sans mémoire $\Omega(K, M)$ correspond à l'espace des états du réseau $\mathcal{S}$. En effet,

$$\Omega(K, M) = \{(x_1, \dots, x_K) \mid \sum_{k=1}^{K} x_k = M \text{ avec } \forall k \leq K\ x_k \leq C_k\} = \mathcal{S}.$$

On peut donc encoder tout sous-ensemble $S \subseteq \mathcal{S}$ dans un diagramme et appliquer les résultats du chapitre 2. Toujours dans ce cadre, la fonction $g$ qui transforme un état en un ensemble d'arcs est donnée par

$$\begin{array}{rccc} g & : & \mathcal{S} & \to & \mathcal{P}(\mathcal{A}) \\ & & \mathbf{x} & \mapsto & \bigcup_{k=1}^{K} \left\{ \left((k-1, \sum_{i=1}^{k-1} x_i), (k, \sum_{i=1}^{k} x_i)\right) \right\}, \end{array} \tag{3.1}$$

et la valeur d'un arc $a = \left((k-1, \ell), (k, \ell')\right) \in A$ par

$$v(a) := \{m \in \mathcal{E}_k \text{ tel que } f^{(k-1)} = \ell \text{ et } f^{(k)} = \ell'\}.$$

Comme les fonctions $f^{(k)}$ correspondent à des sommes cumulées, on en déduit que $v(a) = \{\ell' - \ell\}$, ainsi pour tout $b \in \mathcal{A}$ on a $|v(b)| = 1$ on note alors $v(a) = \ell' - \ell$. Le diagramme complet est donné par $\mathcal{D}(K, M) = (\mathcal{N}, g(\mathcal{S}))$. Dans la suite de cette section, on note $\mathcal{D}$ le diagramme complet. Pour tout $\mathbf{x} \in \mathcal{S}$, on a $f^{(0)}(\mathbf{x}) = 0$, ainsi chaque diagramme a pour nœud source $(0, 0)$ et pour nœud destination $(K, M)$. Considérons maintenant un diagramme $D = (\mathcal{N}, A)$.

**Exemple 11.** *Reprenons l'espace d'états $\mathcal{S}$ défini dans l'exemple 10 ($K = 5$, $M = 3$ et $\mathbf{C} = (2, 1, 3, 1, 2)$). Le diagramme complet correspondant est représenté en figure 3.2. Il contient $|A| = 28$ arcs. Ainsi avec 28 arcs on peut représenter $|\mathcal{S}| = 23$ états. L'ensemble des arcs contenus dans $g(\mathbf{x})$ pour $\mathbf{x} = (0, 0, 2, 0, 1)$ est tracé en pointillé et rouge dans la figure 3.2. L'arc $\left((2, 0), (3, 2)\right)$ a pour valeur $v(a) = 2 - 0 = 2 = x_3$. Graphiquement cela correspond à sa pente.*

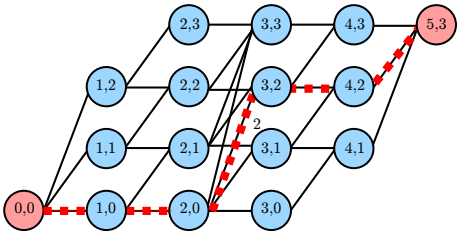



**FIGURE 3.2:** Diagramme complet pour $K = 5$, $M = 3$, et $\mathbf{C} = (2, 1, 3, 1, 2)$.

Dans ce petit exemple, nous ne voyons pas les avantages de la représentation par diagramme. Considérons maintenant les paramètres suivant $K = 25$, $M = 100$ et $\mathbf{C} = (10, \dots, 10)$, on a alors $|A| = 15400$ et $|\mathcal{S}| = 7.9\ 10^{23}$.

### 3.2.2 Transition dans les diagrammes

#### Définition

Dans le chapitre 2, nous avons défini les fonctions de transformation d'un sous-ensemble d'états $S$ en un diagramme $D$ et réciproquement par

$$\phi(S) := (\mathcal{N}, g(S)) \qquad \text{et} \qquad \psi(D) := \bigcup_{S \subseteq \Omega(K,M) \text{ tel que } A = g(S)} S.$$

Dans la section 3.1.1 on a défini $t_{i,j,s}$ la fonction qui modélise le routage d'un client de la file $i$ vers la file $j$ quand le nombre de clients en file $i$ est au moins égal à $s$. On s'intéresse maintenant à $T_{i,j,s}$ la fonction analogue à $t_{i,j,s}$ dans un diagramme. Définissons-la à partir des fonctions de transformation en posant

$$T_{i,j,s} = \phi \circ t_{i,j,s} \circ \psi. \tag{3.2}$$

La définition de $T_{i,j,s}$ assure que pour tout état $\mathbf{x} \in S$ "contenu" dans le diagramme $D = (\mathcal{N}, g(S))$, le diagramme $T_{i,j,s}(D)$ "contient" lui aussi l'état $t_{i,j,s}(\mathbf{x})$.

**Exemple 12.** *Considérons une nouvelle fois l'exemple 10. Dans la figure ci-dessous, les états* $\mathbf{x} = (0,0,2,0,1) \in \mathcal{S}$ *et* $t_{3,1,2}(\mathbf{x}) = (1,0,1,0,1) \in \mathcal{S}$ *sont tracés en pontillé et rouge.*

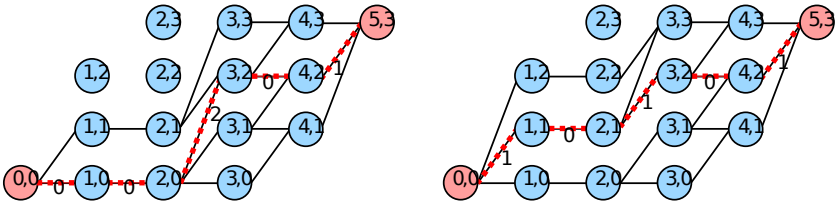



**FIGURE 3.3:** Transition $T_{3,1,2}$.

**Lemme 9.** *Pour un ensemble d'états* $S \subseteq \mathcal{S}$*, un diagramme* $D$*, un couple de files* $(i,j) \in \mathcal{Q}^2$*, et* $s \in \{1, \dots, E_i\}$*, les propriétés suivantes sont satisfaites :*

*i) Si* $S \subseteq \psi(D)$ *alors* $t_{i,j,s}(S) \subseteq \psi(T_{i,j,s}(D))$*,*

*ii) Si* $|\psi(D)| = 1$ *alors* $|\psi(T_{i,j,s}(D))| = 1$*.*

*Démonstration.* *i)* Puisque $S \subseteq \psi(D)$ on a $t_{i,j,s}(S) \subseteq t_{i,j,s}(\psi(D))$. Pour tout $S' \subseteq \mathcal{S}$ nous avons $S' \subseteq \psi \circ \phi(S')$, ainsi $t_{i,j,s}(S) \subseteq \psi \circ \phi \circ t_{i,j,s}(S) \subseteq \psi \circ \phi \circ t_{i,j,s}(\psi(D)) = \psi(T_{i,j,s}(D))$ (Lemme 5).

*ii)* Si $|\psi(D)| = 1$ alors $|t_{i,j,s}(\psi(D))| = 1$. Notons $S' = t_{i,j,s}(\psi(D))$. Puisque $|S'| = 1$, le diagramme $\phi(S')$ ne contient qu'un seul chemin et $|\psi \circ \phi(S')| = 1$.

□

Pour réaliser la transition avec la formule (3.2) on doit calculer un ensemble d'états puis réaliser la transition pour tous les états. Or on a vu que l'espace des états a une cardinalité en $O(K^M)$.

En pratique, la transformation $T_{i,j,s}$ est donc réalisée directement sur le diagramme, sans utiliser les fonctions $t_{i,j,s}$, $\psi$ et $\phi$ avec une complexité en $O(KM^2)$ à l'aide de l'algorithme faisant l'objet du paragraphe suivant.

### Algorithme

On présente maintenant un algorithme qui prend pour entrée un diagramme $D \subseteq \mathcal{D}(K, M)$, un triplet $(i, j, s)$ et retourne $T_{i,j,s}(D)$ avec une complexité en $O(KM^2)$. Pour comprendre comment l'algorithme fonctionne, regardons ce qui se passe quand nous appliquons $t_{i,j,s}$ sur un sous-ensemble d'états $S \subseteq \mathcal{S}$. On a

$$t_{i,j,s}(\mathbf{x}) = \mathbf{x} + (\mathbf{e_j} - \mathbf{e_i})\mathbb{1}_{\{x_i \geq s \text{ et } x_j < C_j\}}.$$

L'ensemble $S$ peut donc être séparé en deux sous-ensembles disjoints $S_1$ et $S_2$ correspondant respectivement à l'ensemble des états $\mathbf{x} \in S$ tels que $t_{i,j,s}(\mathbf{x}) = \mathbf{x}$ et à celui tel que $t_{i,j,s}(\mathbf{x}) \neq \mathbf{x}$. Ainsi, $S_1 = \{\mathbf{x} \in S \mid x_i < s \text{ ou } x_j = C_j\}$ et $S_2 = \{\mathbf{x} \in S \mid x_i \geq s \text{ et } x_j < C_j\}$. Ces sous-ensembles forment une partition de $S$. De plus, ils sont tels que $t_{i,j,s}(S) = S_1 \cup t_{i,j,s}(S_2)$.

On cherche maintenant à identifier les ensembles d'arcs correspondant à $S_1$ et $S_2$ dans le diagramme $D$. Posons $S = \psi(D)$ et $S = S_1 \cup S_2$. Les arcs du diagramme $D$ peuvent être séparés en trois sous-ensembles $\mathcal{S}tay$, $\mathcal{F}ull$ et $\mathcal{T}ransit$ tels que $g(S_1) = \mathcal{S}tay \cup \mathcal{F}ull$ et $g(S_2) = \mathcal{T}ransit$, où $g$ est défini par l'équation 3.1. O a alors :

- $\mathcal{S}tay = \bigcup_{\{\mathbf{x} \in \psi(D) \mid x_i < s\}} g(\mathbf{x})$ ;
- $\mathcal{F}ull = \bigcup_{\{\mathbf{x} \in \psi(D) \mid x_j = C_j\}} g(\mathbf{x})$ ;
- $\mathcal{T}ransit = \bigcup_{\{\mathbf{x} \in \psi(D) \mid x_i \geq s \text{ et } x_j < C_j\}} g(\mathbf{x})$.

Les ensembles $\mathcal{S}tay$, $\mathcal{F}ull$ et $\mathcal{T}ransit$ et ne sont pas forcément disjoints (voir figure 3.4). On identifie les ensembles $\mathcal{S}tay$, $\mathcal{F}ull$ et $\mathcal{T}ransit$ directement dans le diagramme en utilisant les valeurs des arcs et la notion de chemin dans un diagramme. En effet, la valeur d'un arc $a = \big((k-1, \ell), (k, \ell')\big) \in g(\mathbf{x})$ est telle que $v(a) = \ell' - \ell = x_k$. Ainsi pour identifier les arcs contenus dans $\mathcal{S}tay$, on commence par sélectionner dans la colonne $i$ les arcs $a \in A(i)$ tels que $v(a) < s$. Ces arcs correspondent aux états tels que $x_i < s$. Une fois ces arcs identifiés on sélectionne tous les arcs pouvant être reliés à un arc en colonne $i$ tel que $v(a) < s$. En utilisant la définition de $\mathcal{P}ath(B)$ définie dans le chapitre 2 (section 2.2.5) pour $B \subseteq \mathcal{A}$, on obtient ainsi

- $\mathcal{S}tay = \mathcal{P}ath(A_{i,\mathcal{S}tay})$ avec $A_{i,\mathcal{S}tay} = \{a \in A(i) \mid v(a) < s\}$ ;
- $\mathcal{F}ull = \mathcal{P}ath(A_{j,\mathcal{F}ull})$ avec $A_{j,\mathcal{F}ull} = \{a \in A(j) \mid v(a) = C_j\}$ ;
- $\mathcal{T}ransit = \mathcal{P}ath(A(i) \setminus A_{i,\mathcal{S}tay}) \cap \mathcal{P}ath(A(j) \setminus A_{j,\mathcal{F}ull})$.

Maintenant que les sous-ensembles d'arcs $\mathcal{S}tay$, $\mathcal{F}ull$ et $\mathcal{T}ransit$ sont identifiés, on s'intéresse à la manière de transformer $\mathcal{T}ransit$. Regardons ce qui se passe pour $\mathbf{x} \in S_2$ et $\mathbf{y} = t_{i,j,s}(\mathbf{x})$ (*i.e* $\mathbf{x} \neq \mathbf{y}$ ).

$$y_k = \begin{cases} x_k - 1 & \text{si } k = i \\ x_k + 1 & \text{si } k = j \\ x_k & \text{sinon.} \end{cases}$$

Numérotons les arcs contenus dans $g(\mathbf{x})$ et $g(\mathbf{y})$ en les considérant comme des chemins de gauche à droite, ainsi $\{a_1, a_2, \ldots, a_K\} = g(\mathbf{x})$ et $\{b_1, b_2, \ldots, b_K\} = g(\mathbf{y})$.

Soit $a_k = \big((k-1, \ell), (k, \ell')\big) \in g(\mathbf{x})$, on a $v(a_k) = \ell' - \ell = x_k$. Dans chaque colonne $k$, on construit $b_k$ afin que $v(b_k) = y_k$ tout en veillant à conserver la cohérence du chemin $g(\mathbf{y})$. Cette construction dépend du signe de $j - i$. On distingue donc deux cas.

**Cas $i < j$.**

$$b_k = \begin{cases} a_k & \text{si } k < i \\ \big((i-1,\ell),(i,\ell'-1)\big) & \text{si } k = i \\ \big((k-1,\ell-1),(k,\ell'-1)\big) & \text{si } i < k < j \\ \big((j-1,\ell-1),(j,\ell')\big) & \text{si } k = j \\ a_k & \text{si } k > j \end{cases}$$

**Cas $i > j$.**

$$b_k = \begin{cases} a_k & \text{si } k < j \\ \big((j-1,\ell),(j,\ell'+1)\big) & \text{si } k = j \\ \big((k-1,\ell+1),(k,\ell'+1)\big) & \text{si } j < k < i \\ \big((i-1,\ell+1),(i,\ell')\big) & \text{si } k = i \\ a_k & \text{si } k > i \end{cases}$$

On ne modifie ainsi que les arcs des colonnes ayant des numéros compris entre $i$ et $j$ inclus. La connexion avec les colonnes non modifiées reste inchangée.

**Exemple 13.** *Considérons de nouveau l'ensemble d'état $\mathcal{S}$ de l'exemple 10 ($K = 5$, $M = 3$ et $\mathbf{C} = (2,1,3,1,2)$). La figure ci-dessous illustre les différents ensembles $\mathcal{S}tay$, $\mathcal{F}ull$, $\mathcal{T}ransit$ et $\mathcal{T}ransit'$ pour le diagramme $D$ et la transition $T_{4,2,1}$.*

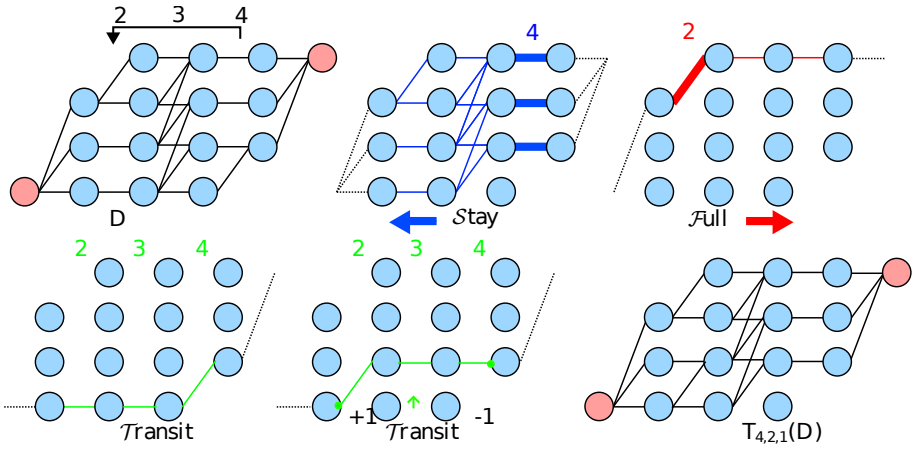



**FIGURE 3.4:** Transition $T_{4,2,1}$.

La complexité de cet algorithme est liée au nombre d'arcs du diagramme $D$ à prendre en compte pour effectuer la transition $T_{i,j,s}$. En effet, trouver les sous-ensembles ($\mathcal{S}tay$, $\mathcal{F}ull$, $\mathcal{T}ransit$) et transformer les arcs de $\mathcal{T}ransit$ se fait en effectuant $|A(i)| + \ldots + |A(j)|$ opérations. Comme $|i-j| \leq K$ et $|A(k)| = O(M^2)$ on en en déduit que la complexité de l'algorithme de transition est en $O(KM^2)$.

### 3.2.3 Simulation parfaite

Maintenant que nous disposons d'un algorithme pour effectuer les transitions sur les diagrammes on peut réadapter l'algorithme PSS (algorithme 10) en utilisant non plus des ensembles états mais des diagrammes.

On note $(U_{-n})_{n\in\mathbb{N}}$ une suite de variable aléatoire i.i.d. telle que $\mathbb{P}(U_0 = (i,j,s)) = \frac{\mu_i p_{i,j}}{\sum_{k\in Q} E_k \mu_k}$.

**Algorithme 11:** $T_{i,j,s}$

**Données** : $D \subseteq \mathcal{D}(K, M)$ un diagramme
**Résultat** : $T_{i,j,s}(D)$

**début**
  $\mathcal{S}tay \leftarrow \mathcal{P}ath(A_{i,\mathcal{S}tay})$;
  $\mathcal{F}ull \leftarrow \mathcal{P}ath(A_{j,\mathcal{F}ull})$ ;
  $\mathcal{T}ransit \leftarrow \mathcal{P}ath(A(i) \setminus A_{i,\mathcal{S}tay}) \cap \mathcal{P}ath(A(j) \setminus A_{j,\mathcal{F}ull})$ ;
  $\mathcal{T}ransit' = \emptyset$ ;
  **si** $i < j$ **alors**
    **pour** $a = \big((k-1,\ell),(k,\ell')\big) \in \mathcal{T}ransit$ **faire**
      **si** $k == i$ **alors**
        $\mathcal{T}ransit' \leftarrow \mathcal{T}ransit' \cup \{\big((i-1,\ell),(i,\ell'-1)\big)\}$ ;
      **si** $i < k < j$ **alors**
        $\mathcal{T}ransit' \leftarrow \mathcal{T}ransit' \cup \{\big((i-1,\ell-1),(i,\ell'-1)\big)\}$ ;
      **si** $k == j$ **alors**
        $\mathcal{T}ransit' \leftarrow \mathcal{T}ransit' \cup \{\big((i-1,\ell-1),(i,\ell')\big)\}$ ;
  **si** $i > j$ **alors**
    **pour** $a = \big((k-1,\ell),(k,\ell')\big) \in \mathcal{T}ransit$ **faire**
      **si** $k == j$ **alors**
        $\mathcal{T}ransit' \leftarrow \mathcal{T}ransit' \cup \{\big((j-1,\ell),(j,\ell'+1)\big)\}$ ;
      **si** $j < k < i$ **alors**
        $\mathcal{T}ransit' \leftarrow \mathcal{T}ransit' \cup \{\big((k-1,\ell+1),(k,\ell'+1)\big)\}$ ;
      **si** $k == i$ **alors**
        $\mathcal{T}ransit' \leftarrow \mathcal{T}ransit' \cup \{\big((i-1,\ell+1),(i,\ell')\big)\}$ ;
  $A' \leftarrow \bigcup_{k<\min(i,j)} A(k) \cup \mathcal{S}tay \cup \mathcal{F}ull \cup \mathcal{T}ransit' \cup \bigcup_{k>\max(i,j)} A(k)$ ;
  **renvoyer** $D' = (\mathcal{N}, A')$.

Pour un temps de simulation $n$ fixé, l'algorithme PSS, applique la transition $T := T_{U_{-1}} \circ \cdots \circ T_{U_{-n}}$ sur le diagramme complet. Cette dernière est déterminée à l'aide de la suite des innovations $(U_{-n})_{n\in\mathbb{N}}$. Si la transition $T$ transforme le diagramme complet en un diagramme ne représentant qu'un seul état (*i.e.* si $|A| = K$) alors l'algorithme se termine et renvoie cet état. Dans le cas contraire, on recommence le procédé en doublant le temps de simulation $n$.

**Algorithme 12:** Simulation parfaite à l'aide de diagrammes (PSD)

**Données** : $(U_{-n} = (i_{-n}, j_{-n}, s_{-n}))_{n\in\mathbb{N}}$ une suite i.i.d de v.a.
**Résultat** : $\mathbf{x} \in \mathcal{S}$

**début**
  $n \leftarrow 1$;
  $T \leftarrow T_{U_{-1}}$;
  $D \leftarrow \psi(T(\mathcal{D}))$ ;
  **tant que** $|A| > K$ **faire**
    $n \leftarrow 2n$;
    $T \leftarrow T_{U_{-1}} \circ \cdots \circ T_{U_{-n}}$;
    $D \leftarrow \psi(T(\mathcal{D}))$ ;
  **renvoyer** $\mathbf{x}$, *l'unique élément de* $\psi(D)$.

**Théorème 6** (Simulation parfaite avec diagrammes)**.** *L'algorithme 12 (PSD) se termine avec probabilité* $1$ *en un temps fini et renvoie* $\mathbf{x} \in \mathcal{S}$ *distribué selon la distribution stationnaire de* $(X_n)_{n\in\mathbb{N}}$.

La plus grande difficulté dans la démonstration du théorème 6 est de montrer qu'il existe une suite de transitions telles que appliquées au diagramme complet ces dernières le transforment en un diagramme ne contenant plus qu'un seul chemin. On considère qu'il existe une suite

finie de transitions $T = T_{i_1,j_1,s_1} \circ \ldots \circ T_{i_N,j_N,s_N}$ tel que $\psi(T(\mathcal{D})) = 1$ et on fera la preuve de ce résultat à la fin de ce chapitre en section 3.5.

*Démonstration.*

**a) L'algorithme se termine**

Faisons l'hypothèse qu'il existe une suite couplante de transitions $T$ (c'est-à-dire $T$ telle que $|\psi(T(\mathcal{D}))| = 1$). Notons $\ell$ la longueur de cette suite. L'algorithme PSD prend pour entrée une suite i.i.d de variables aléatoires $(U_{-n})$. Soit $p_c$ la probabilité que $T_{U_{-1}} \circ \ldots \circ T_{U_{-\ell}} = T$ et $\tau$ le plus petit temps $n$ tel que $|T_{U_{-1}} \circ \ldots \circ T_{U_{-n}}(\mathcal{D})| = 1$. Le $ii)$ du lemme 9 implique que chaque suite de transitions qui contient une suite couplante de transition est-elle même couplante. Ainsi, puisque $(U_{-n})$ est i.i.d. , on a

$$\mathbb{P}(\tau > k \times \ell) \leq (1 - p_c)^k \qquad \text{et} \qquad \mathbb{E}(\tau) \leq \frac{\ell}{p_c}.$$

Par conséquent $\tau$ a une espérance finie. L'algorithme se termine avec probabilité 1. La taille de la suite $(U_{-n})$ utilisée dans l'algorithme est plus petite que $2\ell/p_c$ en moyenne, le facteur 2 vient du doublement de période pour chaque itération de la boucle *tant que* (ligne 5 de l'algorithme 12).

**b) L'algorithme retourne le bon résultat**

Montrons maintenant que PSD renvoie un état distribué selon la distribution stationnaire. Soit $(U_{-n})_{n\in\mathbb{N}}$ une suite i.i.d de variables aléatoires. Comparons les résultats des algorithmes PSS et PSD en utilisant cette même suite. Soit $T_n = T_{U_{-1}} \circ \ldots \circ T_{U_{-n}}$ et $t_n = t_{U_{-1}} \circ \ldots \circ t_{U_{-n}}$, pour tout $n \in \mathbb{N}$. En utilisant un raisonnement par récurrence et le $i)$ du lemme 9 on peut montrer que pour tout $n \in \mathbb{N}$, $t_n(\mathcal{S}) \subseteq T_n(\mathcal{D})$. De plus si, $|t_n(\mathcal{S})| = 1$ alors pour tout $n' \geq n$, $t_{n'}(\mathcal{S}) = t_n(\mathcal{S})$. Notons $\mathbf{y}$ et $\mathbf{x}$ les états respectivement produits par PSS et PSD, nous obtenons donc

$$\{\mathbf{y}\} = t_N(\mathcal{S}) = t_{N'}(\mathcal{S}) \subseteq \psi(T_{N'}(\mathcal{D})) = \{\mathbf{x}\},$$

pour $N$ et $N'$ les tailles des séquences $(U_{-n})_{n\in\mathbb{N}}$ utilisées respectivement par PSS et PSD. Les deux algorithmes retournent le même état, comme l'algorithme PSS retourne un état distribué selon la distribution stationnaire ceci est aussi le cas de PSD. □

## 3.3 Amélioration de la complexité de la représentation

Dans cette section on s'intéresse à des diagrammes pouvant être représentés avec une complexité en $O(KM)$ (au lieu de $O(KM^2)$). Ce sont des diagrammes dits *sans trou*. Nous verrons qu'il est possible de *combler des trous* pour rendre n'importe quel diagramme sans trou quitte à avoir encore une sur représentation de l'ensemble des états "contenus" dans le diagramme.

### 3.3.1 Diagramme sans trou

**Définition 7.** *Un diagramme $D = (\mathcal{N}, A)$ est dit* ***sans trou*** *(Gap Free) si pour tout $l_2$, $v_2$ tels que $l_1 < l_2 < l_3$, $v_1 < v_2 < v_3$, il satisfait les conditions suivantes*

*(GF1)* $\big((k-1,\ell_1),(k,\ell_1+v)\big), \big((k-1,\ell_3),(k,\ell_3+v)\big) \in A \;\Rightarrow \big((k-1,\ell_2),(k,\ell_2+v)\big) \in A;$

*(GF2)* $\big((k-1,\ell),(k,\ell+v_1)\big), \big((k-1,\ell),(k,\ell+v_3)\big) \in A \;\Rightarrow \big((k-1,\ell),(k,\ell+v_2)\big) \in A;$

*(GF3)* $\big((k-1,\ell-v_3),(k,\ell)\big), \big((k-1,\ell-v_1),(k,\ell)\big) \in A \;\Rightarrow \big((k-1,\ell-v_2),(k,\ell)\big) \in A.$

Les conditions $\mathcal{GF}1$, $\mathcal{GF}2$ et $\mathcal{GF}3$ sont illustrées en figure 3.5. La condition $\mathcal{GF}1$ signifie que dans une même colonne l'ensemble des valeurs des arcs forment un intervalle. Les conditions $\mathcal{GF}2$ et $\mathcal{GF}3$ concernent les valeurs des arcs pour chaque nœud. La condition $\mathcal{GF}2$ signifie que pour un même nœud, les valeurs des arcs sortant forment un intervalle et $\mathcal{GF}3$ dit que les arcs entrants forment un intervalle. Remarquons que le diagramme complet ainsi que les diagrammes ne contenant qu'un seul chemin sont sans trou.

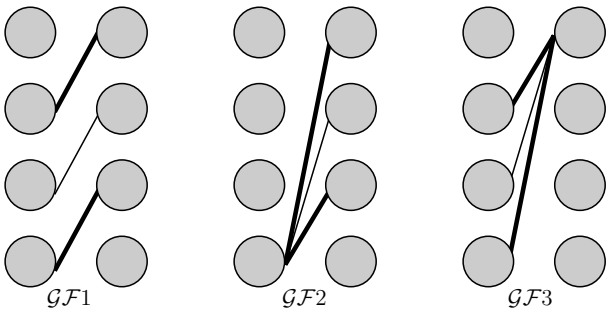



**FIGURE 3.5:** Conditions pour un diagramme sans trou.

Les diagrammes sans trou ont été introduits car on a constaté expérimentalement que l'écrasante majorité des diagrammes successifs utilisés dans PSD sont sans trou. Un contre-exemple de diagramme avec trou obtenu à partir du diagramme complet est illustré en figure 3.4. Le diagramme initial $D$ (qui est sans trou) est obtenu à partir du diagramme complet de la manière suivante $D = T_{4,5,1} \circ T_{4,5,1} \circ T_{2,1,1}(\mathcal{D})$. Comme nous pouvons le constater $D$ est sans trou mais pas $T_{4,2,1}(D)$.

Soit un diagramme $D = (\mathcal{N}, A)$, on note $D^{\mathcal{GF}} = (\mathcal{N}, A^{\mathcal{GF}})$ le diagramme sans trou tel que

$$A^{\mathcal{GF}} = \bigcap_{A' \subseteq \mathcal{A}} \{A' \mid A \subseteq A' \text{ et } D' = (\mathcal{N}, A') \text{ sans trou}\}.$$

Le diagramme $D^{\mathcal{GF}}$ est le diagramme pour lequel on a ajouté le minimum d'arcs possibles au diagramme $D$ pour satisfaire $\mathcal{GF}1$, $\mathcal{GF}2$ et $\mathcal{GF}3$. Si $D$ est déjà sans trou, $D^{\mathcal{GF}} = D$ et on n'ajoute aucun arc supplémentaire. Ainsi $D^{\mathcal{GF}}$ représente au moins autant d'états que $D$. Ceci se traduit par le lemme suivant.

**Lemme 10.** *Soit $D \subseteq \mathcal{D}(K, M)$ un diagramme, alors $\psi(D) \subseteq \psi(D^{\mathcal{GF}})$.*

Bien que contenant plus d'arcs, les diagrammes sans trous peuvent être représentés avec une complexité moindre. Pour chaque colonne, nous n'encodons plus des arcs mais des intervalles de présence d'arcs. Ceci peut ce faire avec une complexité en $O(KM)$, on reviendra plus en détail sur le sujet dans le chapitre 8 qui concerne l'implémentation.

### 3.3.2 Transition dans les diagrammes sans trou

On définit $T^{\mathcal{GF}}_{i,j,s}$ la fonction de transition sur les diagrammes sans trou par

$$T^{\mathcal{GF}}_{i,j,s} = [T_{i,j,s}]^{\mathcal{GF}}.$$

Puisqu'un diagramme sans trou est aussi un diagramme, pour réaliser une transition sur un diagramme $D$ sans trou on commence par calculer $D' = T_{i,j,s}(D)$. Puis on transforme $D'$ en un diagramme sans trou.

Le diagramme complet ainsi que les diagrammes ne contenant qu'un seul chemin sont sans trou. D'après la définition de $T^{\mathcal{GF}}_{i,j,s}$ et le lemme 9 de nous en déduisons le lemme suivant.

**Lemme 11.** *Pour un ensemble d'états $S \subseteq \mathcal{S}$, un diagramme sans trou $D$, $(i,j) \in \mathcal{Q}^2$ et $s \in \{1,\dots,E_i\}$ les propriétés suivantes sont satisfaites :*

*i) Si $S \subseteq \psi(D)$ alors $t_{i,j,s}(S) \subseteq \psi(T^{\mathcal{GF}}_{i,j,s}(D))$,*

*ii) Si $|\psi(D)| = 1$ alors $|\psi(T^{\mathcal{GF}}_{i,j,s}(D))| = 1$.*

Le lemme 11 est l'analogue au lemme 9 pour les diagrammes sans trou.

### 3.3.3 Simulation parfaite pour les diagrammes sans trou

L'algorithme de simulation parfaite en utilisant les diagrammes sans trou est nommé algorithme PSF. Il corresponds à l'algorithme PSD pour lequel on a remplacé les transitions $T_{i,j,s}$ par les transitions sans trou $T^{\mathcal{GF}}_{i,j,s}$.

**Théorème 7** (Simulation parfaite avec diagrammes sans trou)**.** *L'algorithme PSF se termine avec probabilité 1 en un temps d'espérance finie et renvoie $\mathbf{x} \in \mathcal{S}$ distribué selon la distribution stationnaire de $(X_n)_{n\in\mathbb{N}}$.*

Comme dans la section concernant les diagrammes, la plus grande difficulté dans la démonstration du théorème 7 est de montrer qu'il existe une suite de transitions sans trou telle qu'appliquée au diagramme complet cette dernière le transforme en un diagramme ne contenant plus qu'un seul chemin. La preuve se trouve à la fin de ce chapitre en section 3.5. En faisant cette supposition, la démonstration du théorème est en tout point identique à celle du théorème 6 en considérant les résultats du lemme 11 à la place de ceux du lemme 9.

## 3.4 Étude du temps de couplage

Notre intérêt se porte maintenant sur le nombre de transitions qu'il faut effectuer afin que les algorithmes de simulation parfaite fournissent un échantillon distribué selon la distribution stationnaire. On appelle (par abus de langage) **temps de couplage** d'un algorithme de simulation parfaite la valeur prise par l'entier $n$ quand ce dernier retourne l'échantillon. Puisque les transitions sont choisies de manière aléatoire, le temps de couplage est lui aussi aléatoire.

### 3.4.1 Temps de couplage moyen

On note $\mathcal{T}_{PSS}$, $\mathcal{T}_{PSD}$ et $\mathcal{T}_{PSF}$ les variables aléatoires prenant pour valeurs respectives le temps de couplage des algorithmes PSS, PSD et PSF. On s'intéresse aux valeurs moyenne des temps de couplage au travers d'expériences numériques. Puisque pour $S = \psi(D)$, on a

$$t_{i,j,s}(S) \subseteq \psi(T_{i,j,s}(D)) \subseteq \psi(T^{\mathcal{GF}}_{i,j,s}(D)),$$

on en déduit que pour une même suite de variables aléatoires $(U_n)_{n\in Z}$ on a $\mathcal{T}_{PSS} \leq \mathcal{T}_{PSD} \leq \mathcal{T}_{PSF}$. Ce qui nous intéresse ici c'est la différence entre ses valeurs. On étudie deux exemples qui permettent la comparaison avec l'algorithme PSS.

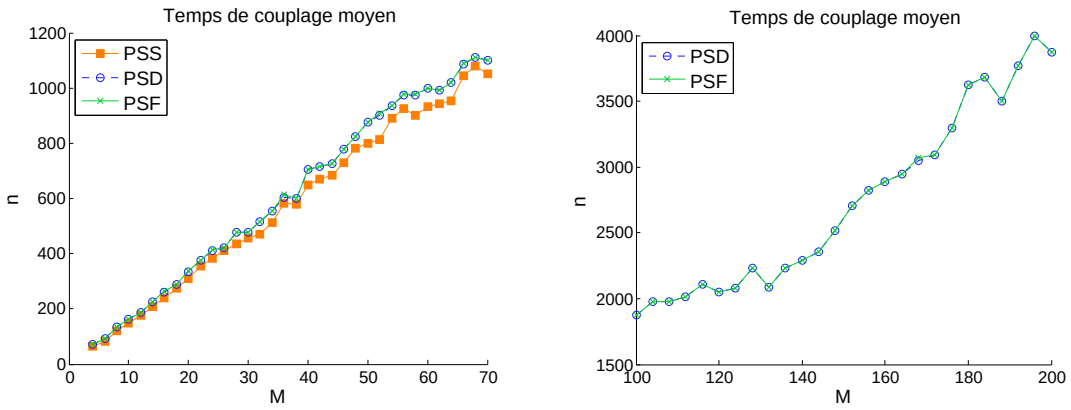



**FIGURE 3.6:** Réseau de l'exemple 10.

### Réseau de l'exemple 10

On considère le réseau de l'exemple 10 en faisant varier le nombre de clients $M$. Comme le réseau contient un petit nombre de files ($K = 5$) ceci limite $|\mathcal{S}|$ (qui est en $O(M^K)$) et donc permet l'utilisation de PSS pour $M \leq 70$. Pour chaque valeur de $M$ on considère 100 suites $(U_{-n})_{n\in\mathbb{N}}$ de variables aléatoires iid. et pour chacune des suites on effectue 3 simulations : une avec PSS, une avec PSD et une autre avec PSF. On prend pour paramètres

- $M \in \{4, 6, \ldots, 70\} \cup \{100, 104, \ldots, 200\}$ (nombre de clients) ;
- $\mathbf{C} = (\frac{M}{2}, \frac{M}{2}, \frac{M}{2}, \frac{M}{2}, \frac{M}{2})$ (capacités des files) ;
- $\mathbf{E} = (2, 2, 2, 2, 2)$ (nombre de serveurs) ;
- $\mu = (1, 1, 1, 1, 1)$ (taux de service).

La figure 3.6 compare les temps de couplage moyens pour les différents algorithmes de simulation parfaite. Dans la figure de gauche on compare PSS, PSD et PSF pour $M \in \{4, 6, \ldots, 70\}$. Pour $M > 70$, le nombre d'états entraîne une explosion de la taille mémoire pour l'encodage de $\mathcal{S}$ et ne nous permet donc pas réaliser des simulations avec PSS. On continue donc la simulation dans la figure de droite, uniquement pour les algorithmes PSD et PSF avec $M \in \{100, 104, \ldots, 200\}$. Les temps de couplage moyen de PSD et PSF se confondent sur les figures. Celui de PSS est légèrement inférieur aux deux autres.

### Réseau en anneau

On considère un réseau en anneau défini tel que $p_{i,j} = 1$ si et seulement si $j = (i + 1) \mod K$. Dans ce réseau particulier, Bouillard *et al.* ont montré dans [14] qu'il existe un ensemble d'états $S_{xt} \subseteq \mathcal{S}$ tel que $|S_{xt}| = K$, appelés **états extrémaux** tel que pour toute suite de transitions $t$ on ait $|t(\mathcal{S})| = 1$ si et seulement si $|t(S_{xt})| = 1$. Ainsi grâce à ce résultat, on peut comparer les temps de couplage de PSS, PSD et PSF pour un plus grand nombre de files et de clients.

- $K \in \{4, 6, \ldots, 20\}$ (nombre de files) ;
- $M = 2K$ (nombre de clients) ;
- $\mathbf{C} = (M, \ldots, M)$ (capacités des files) ;
- $\mathbf{E} = (2, 2, 2, 2, 2)$ (nombre de serveurs) ;
- $\mu = (1, 1, 1, 1, 1)$ (taux de service).

La figure 3.7 compare les temps de couplage pour les différents algorithmes de simulation parfaite. Dans la figure de droite on a refait la même expérimentation avec les mêmes

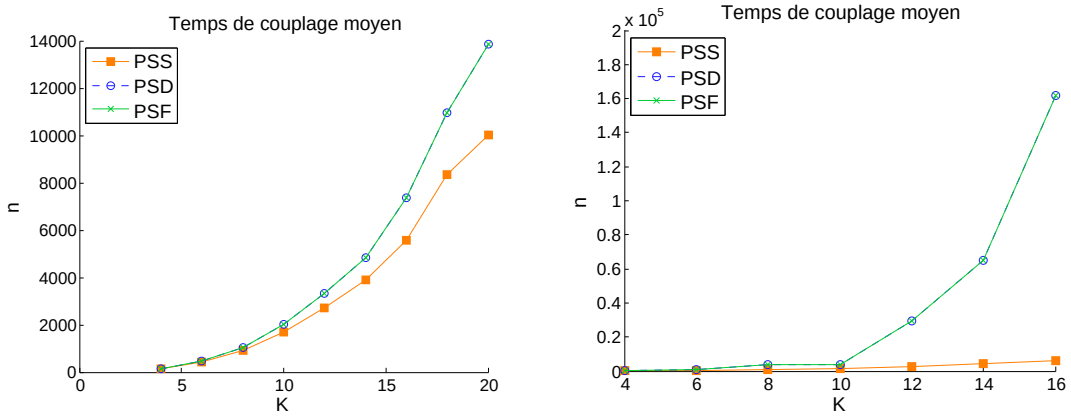



**FIGURE 3.7:** Réseau en anneau.

paramètres mais en renumérotant pour chaque simulation les files à l'aide de permutations choisies aléatoirement. Les temps de couplage moyen de PSD et PSF, explosent en comparaison de celui de PSS dans la figure de droite, tandis qu'ils semblent assez proche dans la figure de gauche.

Pour un diagramme $D$ et un ensemble d'états $S$ tels que $S = \psi(D)$, plus le nombre $|i-j|$ est grand, plus l'ensemble $T_{i,j}(D) \setminus t_{i,j}(S)$ est grand. Ainsi, pour favoriser des temps de couplage pour PSD et PSF proches de celui de PSS, il faut minimiser la fréquence d'apparition des transitions $T_{i,j}$ avec $|i-j|$ grand. C'est ce que nous allons expérimenter dans la sous-section suivante.

### 3.4.2 Numérotation des files

Plus la valeur absolue $|j-i|$ est grande, moins la transition $T_{i,j,s}$ est précise. Pour trouver un bon ordre pour les files, on propose d'évaluer chaque numérotation de files. On représente chaque numérotation par une permutation $\sigma$ de longueur $K$ qu'on évalue par la fonction $e$ à valeur dans $\mathbb{R}$ définie par

$$e(\sigma) = \sum_{i,j \in \mathcal{Q}^2,\ k=\sigma(i), k'=\sigma(j)} (\mu_i\ E_i\ p_{i,j} + \mu_j\ E_j\ p_{j,i})(k-k')^2.$$

La numérotation initiale correspond à l'identité. On s'intéresse au temps de couplage en fonction de la valeur $e(\sigma)$. Pour plus effectuer la simulation plusieurs fois avec la même suite d'innovation, on considère le temps de couplage vers l'avant, c'est-à-dire sans doublement de période et en commençant les simulations au temps $0$ jusqu'à ce que les trajectoires couplent.

On effectue les simulations à la fois sur l'ensemble des états et sur le diagramme. On débute chaque simulation au temps $0$ en prenant $S = \mathbb{S}$ et $D = \mathcal{D}$. À chaque pas de simulation $n$, on tire aléatoirement un triplet $(i, j, s)$ (selon les paramètres du réseau) et on applique les transitions : $S = t_{i,j,s}(S)$ et $D = T_{i,j,s}(D)$. La simulation s'arrête quand $|\psi(T(\mathcal{D}))| = 1$.

**Réseau de l'exemple 10.** On considère les paramètres de l'exemple 10 ($K = 5$ files) en prenant pour paramètres :

- $M = 20$, nombre de clients ;
- $\mathbf{C} = (M/2, \ldots, M/2)$, capacités des files ;
- $\mathbf{E} = (1, 1, 1, 1, 1)$, nombre de serveurs ;

– $\mu = (3, 1, 1, 1, 1)$,taux de service.

Pour les paramètres ci-dessus, on dénombre $80$ évaluations différentes à valeur dans l'intervalle $[16.85, 68.35]$. La permutation initiale c'est-à-dire $\sigma = Id$, a pour évaluation $e(Id) = 25.4$. La permutation $(1, 3, 2, 4, 5)$ obtient le meilleur score qui est de $16.85$, elle correspond à la meilleure numérotation du réseau et est illustrée en figure 3.8. Le plus mauvais score est obtenu par les permutations $(1, 4, 5, 3, 2)$ et $(5, 2, 1, 3, 4)$. Pour la simulation, on ordonne les permutations en fonction de leurs évaluations, de la plus petite à la plus grande et on ne retient qu'une permutation sur $4$. La figure 3.9 illustre les temps de couplage moyen pour les ensembles d'états et les diagrammes. Chaque point correspond à une permutation pour laquelle on a effectué $2000$ simulations et calculé la moyenne. On constate que pour une évaluation $e(\sigma)$ petite, le temps de couplage de la simulation qui utilise les ensembles d'états est assez proche du temps de couplage pour celle qui utilise les diagrammes. Cette expérimentation est conforme à notre intuition.

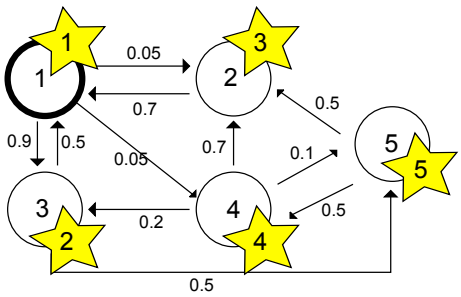



**FIGURE 3.8:** Meilleure permutation.

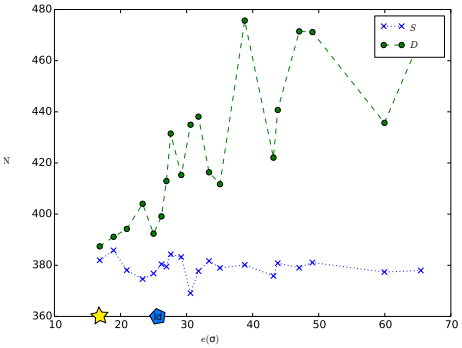



**FIGURE 3.9:** Temps de couplage moyen.

Trouver la permutation minimisant $e(\sigma)$ s'apparente au problème du voyageur de commerce (qui est NP-complet). Ici, nous ne cherchons pas forcément à trouver la permutation optimale mais une permutation donnant un temps de couplage pas trop mauvais. La manière la plus simple, est de procéder au tirage aléatoire d'un nombre fixé $N$ de permutations et de retenir la permutation $\sigma$ optimisant $e$. Plus ce nombre sera grand, plus la permutation retenue sera optimale. Cependant nous ne possédons pas de résultat qui permettrait de choisir $N$ en fonction de la précision souhaitée.

### 3.4.3 Technique de *Splitting*

Les transitions sur un ensemble des états s'opèrent de manière exacte ce qui n'est pas le cas pour les diagrammes. En effet pour $S = \psi(D)$ on a $t_{i,j,s}(S) \subseteq \psi(T_{i,j,s}(D))$. L'idée est alors de détecter le $n$ pour lequel $|\psi(T_n(\mathcal{D}))|$ est assez petit pour pouvoir continuer la simulation avec l'ensemble d'états $\psi(T_n(\mathcal{D}))$ au lieu du diagramme $T_n(\mathcal{D})$. Cette idée fut introduite sous le nom de *splitting* dans [18].

**Réseau de l'exemple 10.** Nous reprenons les paramètres de la simulation précédente en augmentant le nombre de clients et en la capacité de chaque file. On prend $M = 50$ et $\mathbf{C} = (M, \ldots, M)$. L'espace des états contient $|\mathcal{S}| = 316251$ états et le diagramme complet $|\mathcal{A}| = 4080$ arcs.

La figure 3.10 illustre une simulation vers l'avant. Au temps 0, on a $S = \mathcal{S}$ (l'espace des états) et $D = \mathcal{D}$ (le diagramme complet). Pour chaque pas de simulation $n$, on tire un triplet $(i, j, s)$ et on applique les transitions sur l'ensemble d'états $S$ et sur le diagramme $D$. Pour chaque $n$ on peut lire en ordonnée (sur une échelle logarithmique) le nombre d'états $|S|$ et le nombre d'états que représente le diagramme $|\psi(D)|$.

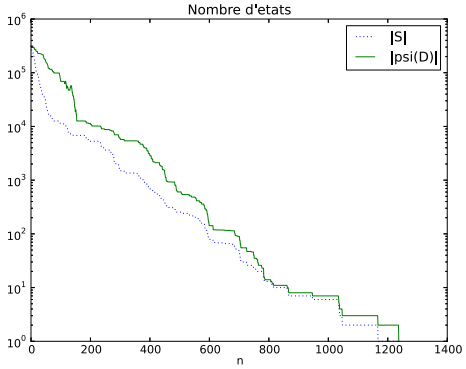



**FIGURE 3.10:** Nombre d'états lors de la simulation d'un couplage vers l'avant.

| Moyenne du temps de couplage | | | | Moyenne des temps de survenue du *split* |
|---|---|---|---|---|
| $M$ | États ($S$) | *Split* ($D$ puis $\psi(D)$) | Diagramme ($D$) | (premier $n$ tel que $\lvert\psi(D)\rvert < 1000$) |
| 10 | 191.39 | 191.39 | 198.632 | 1.0 |
| 20 | 385.558 | 394.7 | 397.69 | 60.216 |
| 30 | 557.214 | 569.562 | 570.66 | 140.954 |
| 40 | 736.418 | 752.396 | 754.238 | 242.744 |
| 50 | 904.198 | 920.99 | 923.022 | 349.242 |
| 60 | - | 1089.83 | 1091.344 | 456.412 |
| 70 | - | 1282.26 | 1283.08 | 583.632 |
| 80 | - | 1438.414 | 1439.238 | 702.212 |
| 90 | - | 1608.22 | 1609.202 | 834.762 |
| 100 | - | 1765.086 | 1765.802 | 973.762 |

**TABLE 3.1:** Comparaison des temps de couplages moyens.

Le couplage de l'espace des états intervient au temps $n = 1167$ et celui pour du diagramme complet au temps $n = 1237$. Cette simulation a due être effectuée une dizaine de fois avant d'obtenir des temps de couplages différents pour $S$ et $D$. Notons aussi que la différence des temps de couplage n'est pas énorme, elle ne correspond qu'à $5\%$ du temps de simulation total. Cependant en cas de doublement de période comme nous le faisons pour les algorithmes `PSS`, `PSD` et `PSF`, un petit décalage de temps couplage peut entraîner le doublement du temps de simulation.

Dans l'expérimentation illustrée par le tableau 3.1, on fait varier le nombre de clients $M \in \{10, 20, \ldots, 100\}$ et on considère des files de capacité infinies ($\mathbf{C} = (M, \ldots, M)$). On effectue les simulations vers l'avant à la fois sur les états et sur le diagramme. Quand le nombre d'états représenté par un diagramme est inférieur ou égal à un seuil que l'on a fixé à $|\psi(D)| = 1000$, on transforme le diagramme en un ensemble d'état puis on continue la simulation avec l'ensemble des états. Ce type de simulation est appelée *Split* dans le tableau. On s'intéresse à la moyenne des temps de couplage par l'avant pour $500$ simulations. Pour $M > 50$, le trop grand nombre d'états dans l'espace des états ne nous permet pas d'utiliser la simulation avec les ensembles d'états. La dernière colonne donne la moyenne du temps $n$ pour lequel on a effectué la transformation du diagramme en un ensemble d'états. Comme on peut le constater, le gain en temps de simulation en utilisant *Split* est négligeable en comparaison du temps de simulation en utilisant les diagrammes. Il est le meilleur pour $M = 50$ et $M = 60$ puis il décroît quand le nombre de clients augmente.

## 3.5 Preuve d'existence d'une suite couplante de transitions

Afin de prouver le bon fonctionnement de l'algorithme PSD, on a supposé qu'il existait une suite de transitions telles qu'appliquées au diagramme complet cette dernière le transforme en un diagramme ne contenant plus qu'un seul chemin. Nous avons fait la même supposition concernant l'algorithme PSF et les transitions "sans trou". Cette section est dédiée à la preuve de ces deux résultats. Ils peuvent être prouvés en ne considérant que des transitions $T_{i,j,s}$ et $T^{\mathcal{GF}}_{i,j,s}$ pour lesquelles $s = 1$. Pour simplifier les notations, on notera $T_{i,j,1} = T_{i,j}$ et $T^{\mathcal{GF}}_{i,j,1} = T^{\mathcal{GF}}_{i,j}$.

### 3.5.1 Définitions

#### Rappels sur les transitions

Pour effectuer la transition $T_{i,j}$ sur un diagramme $D = (\mathcal{N}, A)$, on sépare $A$ en trois sous-ensembles $\mathcal{S}tay$, $\mathcal{F}ull$ et $\mathcal{T}ransit$ (non nécessairement disjoints) tels que

- $\mathcal{S}tay = \mathcal{P}ath(A_{i,\mathcal{S}tay})$ avec $A_{i,\mathcal{S}tay} = \{a \in A(i) \mid v(a) = 0\}$ (car ici $s = 1$),
- $\mathcal{F}ull = \mathcal{P}ath(A_{j,\mathcal{F}ull})$ avec $A_{j,\mathcal{F}ull} = \{a \in A(j) \mid v(a) = C_j\}$,
- $\mathcal{T}ransit = \mathcal{P}ath(A(i) \setminus A_{i,\mathcal{S}tay}) \cap \mathcal{P}ath(A(j) \setminus A_{j,\mathcal{F}ull})$.

Une fois les sous-ensembles d'arcs identifiés, on construit $\mathcal{T}ransit'$ à partir des arcs de $\mathcal{T}ransit$ en fonction du signe de $i - j$. Par exemple, si $i < j$, alors l'arc $b_k$ est construit à partir de l'arc $a_k = \big((k-1,\ell),(k,\ell')\big) \in A(k)$ de la manière suivante,

$$b_k = \begin{cases} a_k & \text{si } k < i \\ \big((i-1,\ell),(i,\ell'-1)\big) & \text{si } k = i \\ \big((k-1,\ell-1),(k,\ell'-1)\big) & \text{si } i < k < j \\ \big((j-1,\ell-1),(j,\ell')\big) & \text{si } k = j \\ a_k & \text{si } k > j. \end{cases}$$

Pour finir, l'algorithme retourne le diagramme $D' = (\mathcal{N}, \mathcal{S}tay \cup \mathcal{F}ull \cup \mathcal{T}ransit')$.

On introduit maintenant de nouvelles définitions : la propriété d'arbre, le chemin couvrant, le diagramme saturé et la colonne $m$-complète, ainsi que leurs propriétés.

#### La propriété d'arbre

Pour $D = (\mathcal{N}, A)$ diagramme, $k \in \mathcal{Q}$ et $\ell \in \{0, \dots, M\}$ on définit

$$V(k,\ell) := \{v \mid \big((k-1,\ell),(k,\ell+v)\big) \in A(k)\},$$

l'ensemble de toutes les valeurs des arcs ayant pour origine le nœud $(k-1,\ell)$. Si $V(k,\ell) \neq \emptyset$ alors on pose

$$v_{\min}(k,\ell) = \min(V(k,\ell)) \qquad \text{et} \qquad v_{\max}(k,\ell) = \max(V(k,\ell)).$$

**Définition 8.** *Un diagramme $D$ satisfait la **propriété d'arbre** $\mathcal{T}_c$ si pour tout $k \leq c$ les points suivants sont vérifiés :*

*i) $V(k,\ell)$ est un intervalle,*

*ii) Si* $V(k,\ell) \neq \emptyset$ *et* $V(k,\ell+1) \neq \emptyset$*, alors* $v_{\max}(k,\ell) = v_{\min}(k,\ell+1)$.

**Lemme 12.** *Les nœuds de la forme* $(k,\ell)$ *avec* $k \leq c$ *d'un diagramme vérifiant la* ***propriété d'arbre*** $\mathcal{T}_c$ *possèdent au plus un arc entrant.*

*Démonstration.* Raisonnons par l'absurde. Soit $D = (\mathcal{N}, A)$ un diagramme satisfaisant la **propriété d'arbre** $\mathcal{T}_c$. Supposons qu'il existe un nœud $(k,\ell) \in \mathcal{N}$ avec $k \leq c$ possédant plus d'un arc entrant. Alors le diagramme contient au moins deux arcs de la forme $\big((k-1,\ell_1),(k,\ell)\big) \in A$ et $\big((k-1,\ell_2),(k,\ell)\big) \in A$ avec $\ell_1 < \ell_2$. Notons $v_1 = \ell - \ell_1$ et $v_2 = \ell - \ell_2$, on a alors $v_1 > v_2$, $v_1 \in V(k-1,\ell_1)$ et $v_2 \in V(k-1,\ell_2)$. Ainsi $\max(V(k-1,\ell_1)) > \min(V(k-1,\ell_2))$ ce qui contredit $ii)$ pour la colonne $k$. □

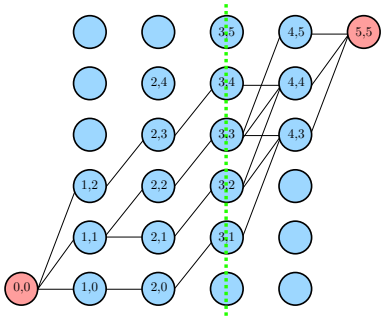



**FIGURE 3.11:** Diagramme $K = 5$, $M = 5$ et $\mathbf{C} = (2,2,1,2,2)$ vérifiant la propriété d'arbre $\mathcal{T}_3$ .

**Lemme 13.** *Si* $D$ *est un diagramme satisfaisant la propriété arbre* $\mathcal{T}_c$ *et si* $i,j \leq c$ *alors* $T_{i,j}(D)$ *satisfait aussi* $\mathcal{T}_c$.

La figure 3.12 illustre la transition $T_{1,3}$ dans un diagramme ayant la propriété $\mathcal{T}_3$. En bleu et gras sont représentés les arcs de l'ensemble $\mathcal{S}tay$ et en rouge et gras ceux de l'ensemble $\mathcal{F}ull$.

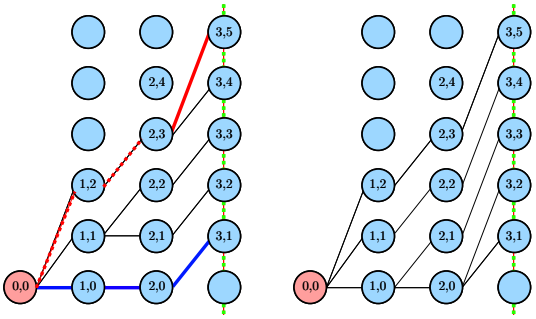



**FIGURE 3.12:** Colonnes 1,2 et 3 des diagrammes $D \subseteq \mathcal{D}(5,5)$ et $D' = T_{1,3}(D)$.

*Démonstration.* On fait l'hypothèse que $i < j$, le raisonnement est identique pour $i > j$. Notons $D' = T_{i,j}(D) = (\mathcal{N}, A')$. Montrons que $D'$ satisfait $\mathcal{T}_c$.

La transition $T_{i,j}$ n'affecte que les colonnes comprises entre $i$ et $j$. Ainsi pour $k \in \{1,\ldots,c\} \setminus \{i,\ldots,j\}$ les points $i)$ et $ii)$ de la définition sont vérifiés pour le diagramme $D'$.

Montrons maintenant que $i)$ et $ii)$ sont vérifiés pour chaque colonne $A'(k)$ avec $k \in \{i, \dots, j\} \subseteq \{1, \dots, c\}$. Posons $\mathcal{S}tay(k) = \mathcal{S}tay \cap A(k)$, $\mathcal{T}ransit(k) = \mathcal{T}ransit \cap A(k)$ et $\mathcal{F}ull(k) = \mathcal{F}ull \cap A(k)$). Comme $A = \mathcal{S}tay \cup \mathcal{T}ransit \cup \mathcal{F}ull$, on a $A(k) = \mathcal{S}tay(k) \cup \mathcal{T}ransit(k) \cup \mathcal{F}ull(k)$.

Si $A(i) = \mathcal{S}tay(i) \cup \mathcal{F}ull(i)$ ou si $A(j) = \mathcal{S}tay(j) \cup \mathcal{F}ull(j)$ alors $D = D'$ et la propriété d'arbre est vérifiée pour $D'$.

Si $\mathcal{S}tay(i) = \mathcal{F}ull(j) = \emptyset$ alors pour chaque colonne $k \in \{i, \dots, j\}$ on a $A(k) = \mathcal{T}ransit(k)$ et $A'(k) = \mathcal{T}ransit'(k)$. Posons

$$V'(k, \ell) = \{v \mid \big((k-1, \ell), (k, \ell + v)\big) \in A'(k)\}.$$

On a alors

- $V'(i, \ell) = \{v - 1 \mid v \in V(k, \ell)\}$ ;
- pour $i < k < j$, $V'(k, \ell) = V(k, \ell + 1)$ ;
- $V'(j, \ell) = \{v + 1 \mid v \in V(k, \ell + 1)\}$.

Pour $k \in \{i, \dots, j\}$, $V(k, \ell)$ est un intervalle, on en déduit ainsi qu'il en est de même pour $V'(k, \ell)$. Par hypothèse, si $V(k, \ell) \neq \emptyset$ et $V(k, \ell + 1) \neq \emptyset$, alors $v_{\max}(k, \ell) = v_{\min}(k, \ell + 1)$. Ainsi si $V'(k, \ell) \neq \emptyset$ et $V'(k, \ell + 1) \neq \emptyset$, alors $v'_{\max}(k, \ell) = v'_{\min}(k, \ell + 1)$. Les conditions $i)$ et $ii)$ sont donc satisfaites pour toute colonne $k \in \{i, \dots, j\}$ du diagramme $D'$.

On s'intéresse maintenant aux cas : $\mathcal{S}tay(i) \neq \emptyset$ et $\mathcal{F}ull(j) \neq \emptyset$. L'idée est dans les deux cas, pour chaque colonne $k \in \{i, \dots, j\}$, de trouver le palier $\tilde{\ell}$ "faisant la frontière" avec $\mathcal{T}ransit(k)$.

Si $\mathcal{S}tay(i) \neq \emptyset$ et $\mathcal{T}ransit(i) \neq \emptyset$. La condition $ii)$ implique que dans la colonne $i$ il existe exactement un nœud $(i-1, \ell)$ tel que $\{0, 1\} \subseteq V(i-1, \ell)$. Notons $\tilde{\ell}$ l'étage de ce nœud. Pour $\ell < \tilde{\ell}$, on a $V(i-1, \ell) = \{0\}$ ou $V(i-1, \ell) = \emptyset$. Pour $\ell > \tilde{\ell}$ on a $0 \notin V(i-1, \ell)$. Comme $D$ a la propriété arbre, les nœuds $(k, \ell)$ ne possèdent qu'un arc entrant. Il y a donc un chemin qui fait la frontière entre les arcs dans $\mathcal{S}tay$ et les autres. De plus pour $k > i$, un nœud a soit tous ses arcs sortant dans $\mathcal{S}tay$ soit tous ses arcs sortants dans $\mathcal{T}ransit$. Soit $(k, \ell) \in \mathcal{N}$, on note $V(k, \ell) \in \mathcal{S}tay$ si les arcs sortant du nœud $(k, \ell)$ appartiennent à l'ensemble $\mathcal{S}tay$. On note $\tilde{\ell}$ le nombre tel que $V(k, \tilde{\ell}) \in \mathcal{S}tay$ et $V(k, \tilde{\ell} + 1) \notin \mathcal{S}tay$. On en déduit ainsi

| Ensemble | $k = i$ | $k > i$ et $k < j$ | $k = j$ |
|---|---|---|---|
| $V'(k, \tilde{\ell} - 1)$ | $V(k, \tilde{\ell} - 1) = \{0\}$ | $V(k, \tilde{\ell} - 1)$ | $V(k, \tilde{\ell} - 1)$ |
| $V'(k, \tilde{\ell})$ | $\{\max(v - 1, 0) \mid v \in V(k, \tilde{\ell})\} \cup \{0\}$ | $V(k, \tilde{\ell}) \cup V(k, \tilde{\ell} + 1)$ | $V(k, \tilde{\ell}) \cup \{v + 1 \mid v \in V(k, \tilde{\ell} + 1)\}$ |
| $V'(k, \tilde{\ell} + 1)$ | $\{v - 1 \mid v \in V(k, \tilde{\ell} + 1)\}$ | $V(k, \tilde{\ell} + 2)$ | $\{v + 1 \mid v \in V(k, \tilde{\ell} + 2)\}$ |

Ainsi pour tout $k \in \{i, \dots, j\}$ on a bien $\max(V'(k, \tilde{\ell} - 1)) = \min(V'(k, \tilde{\ell})$ et $\max(V'(k, \tilde{\ell})) = \min(V'(k, \tilde{\ell} + 1))$. De plus pour $\ell \in \{\tilde{\ell} - 1, \tilde{\ell}, \tilde{\ell} + 1\}$, $V'(k, \ell)$ est un intervalle.

Si $\mathcal{F}ull(j) \neq \emptyset$ et $\mathcal{T}ransit(j) \neq \emptyset$. La condition $ii)$ implique que dans la colonne $j$ il existe exactement un nœud $(j-1, \ell)$ tel que $\{C_j - 1, C_j\} \subseteq V(j-1, \ell)$. Comme $D$ a la propriété d'arbre, les nœuds $(k, \ell)$ ne possèdent qu'un arc entrant. Il y a donc un chemin qui fait la frontière entre les arcs dans $\mathcal{F}ull$ et les autres. Mais comme le sous-ensemble d'arc $\mathcal{T}ransit$ se construit de gauche à droite, pour $k < j$ il existe exactement un arc $a$ tel que $a \in \mathcal{T}ransit(k)$ et $a \in \mathcal{F}ull(k)$. Pour $k < j$, on note $\tilde{\ell}$ le nombre tel que $a = \big((k-1, \tilde{\ell}), (k, \ell')\big) \in \mathcal{T}ransit(k) \cap \mathcal{F}ull(k)$ et pour $k = j$ notons $\tilde{\ell} := \min(\{\ell \mid v(\big((j-1, \ell), (j, \ell')\big))\}) = C_j$.

| Ensemble | $k = j$ | $k < j$ et $k > i$ | $k = i$ |
|---|---|---|---|
| $V'(k, \tilde{\ell}+1)$ | $V(k, \tilde{\ell}+1) = \{C_j\}$ | $V(k, \tilde{\ell}+1)$ | $V(k, \tilde{\ell}+1)$ |
| $V'(k, \tilde{\ell})$ | $\{\min(C_j, v+1) \mid v \in V(k, \tilde{\ell})\} \cup \{C_j\}$ | $V(k, \tilde{\ell})$ | $V(k, \tilde{\ell}) \cup \{v-1 \mid v \in V(k, \tilde{\ell})\}$ |
| $V'(k, \ell-1)$ | $\{v+1 \mid v \in V(k, \tilde{\ell})\}$ | $V(k, \tilde{\ell})$ | $\{v-1 \mid v \in V(k, \ell-1)\}$ |

Ainsi pour tout $k \in \{i, \dots, j\}$ on a bien : $\max(V'(k, \tilde{\ell}-1)) = \min(V'(k, \tilde{\ell})$ et $\max(V'(k, \tilde{\ell})) = \min(V'(k, \tilde{\ell}+1)$. De plus pour $\ell \in \{\tilde{\ell}-1, \tilde{\ell}, \tilde{\ell}+1\}$, $V'(k, \ell)$ est un intervalle.

Les propriétés $i)$ et $ii)$ sont donc vérifiées pour chaque colonne $A'(k)$ avec $k \in \{i, \dots, j\}$.

□

### Chemin couvrant

Dans un réseau fermé de files d'attente, on appelle chemin couvrant tout chemin reliant la file 1 à la file $K$ et passant au moins une fois dans chaque autre file.

**Définition 9.** *Un vecteur $\mathbf{sp} \in \mathcal{Q}^L$ est appelé **chemin couvrant** si :*

- $\mathbf{sp}(1) = 1$ *et* $\mathbf{sp}(L) = K$ *;*
- $\mathcal{Q} = \{\mathbf{sp}(n) \mid 1 \leq n \leq L\}$ *;*
- *si* $i = \mathbf{sp}(n)$ *et* $j = \mathbf{sp}(n+1)$ *alors* $p_{i,j} > 0$ *;*
- *si* $\mathbf{sp}(n) > \mathbf{sp}(n+1)$ *alors il existe* $n' < n$ *tel que* $\mathbf{sp}(n+1) = \mathbf{sp}(n')$.

Comme les réseaux de files d'attentes que nous considérons sont connexes, alors pour chaque réseau il existe toujours au moins un chemin couvrant (quitte à renuméroter les files). On notera $\mathbf{sp}$ un tel chemin et $L$ sa longueur.

### Diagramme saturé

Soit $n \leq L$. On appelle **état** $(\mathbf{sp}, n)$**-saturé** un état dans lequel les clients des files $\mathbf{sp}(n)$, $\mathbf{sp}(n-1)$, …, $\mathbf{sp}(1) = 1$ sont concentrés par ordre de préférence dans les files $\mathbf{sp}(n)$, $\mathbf{sp}(n-1)$, …, $\mathbf{sp}(1)$. Plus formellement, si $\mathbf{x} \in \mathcal{S}$ est $n$-saturé, pour $c = \max(\{\mathbf{sp}(n'), n' \leq n\})$ et $M_c := \sum_{k=1}^{c} x_c = f^{(c)}(\mathbf{x})$ on a

$$x_k = \begin{cases} \max(0, \min(M_c, C_k)) & \text{si } k = \mathbf{sp}(n) \\ \max(0, \min(M_c - \sum_{p=n'+1}^{n} x_{\mathbf{sp}(p)}, C_k)) & \text{si } k = \mathbf{sp}(n') \text{ et } n' < n. \end{cases}$$

**Définition 10.** *On appelle **diagramme** $(\mathbf{sp}, n)$**-saturé** tout diagramme $D$ tel que $\psi(D) \subseteq \mathcal{S}$ soit un ensemble d'états $(\mathbf{sp}, n)$-saturés. Le diagramme $(\mathbf{sp}, n)$-saturé contenant le plus grand nombre d'arcs est appelé **diagramme saturé maximal** et est noté $\mathcal{D}_{(\mathbf{sp},n)}$.*

**Exemple 14.** *Considérons le réseau de la figure 3.1, $K = 5$, $\mathbf{C} = (2, 2, 2, 2, 2)$, et $M = 5$. Le vecteur $\mathbf{sp} = (1, 2, 3, 1, 4, 5)$ est un chemin couvrant pour ce réseau. Pour $n = 4$, le sous-chemin à considérer est $(1, 2, 3, 1)$ et $c = 3$. La figure 3.13 illustre le diagramme $\mathcal{D}_{(\mathbf{sp},4)}$ et les états contenus dans $\psi(\mathcal{D}_{(\mathbf{sp},4)})$ sont :*

$(2 \quad 1 \quad 2 \quad \cdots)$
$(2 \quad 0 \quad 2 \quad \cdots)$
$(2 \quad 0 \quad 1 \quad \cdots)$
$(2 \quad 0 \quad 0 \quad \cdots)$
$(1 \quad 0 \quad 0 \quad \cdots)$

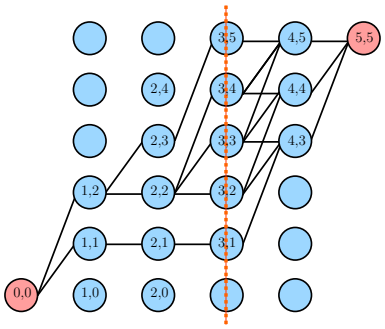



**FIGURE 3.13:** Diagramme $\mathcal{D}_{(\mathbf{sp},4)}$.

Remarquons que $\mathcal{S} = \psi(\mathcal{D})$ est un ensemble d'états $(\mathbf{sp}, 1)$-saturés. Le diagramme complet est donc le diagramme $(\mathbf{sp}, 1)$-saturé maximal ($\mathcal{D} = \mathcal{D}_{(\mathbf{sp},1)}$). De plus, remarquons qu'il n'existe qu'un seul état $\mathbf{x}$ état étant $(\mathbf{sp}, L)$-saturé, ainsi il n'existe qu'un seul diagramme $D$ étant $(\mathbf{sp}, L)$-saturé.

**Lemme 14.** *Soit $\mathcal{D}_{(\mathbf{sp},n)}$ le diagramme $(\mathbf{sp}, n)$-saturé maximal et $c = \max(\{\mathbf{sp}(n'), n' \leq n\})$. Alors $\mathcal{D}_{(\mathbf{sp},n)}$ possède la propriété arbre $\mathcal{T}_c$.*

*Démonstration.* Soit $D = \mathcal{D}_{(\mathbf{sp},n)}$ le diagramme $(\mathbf{sp}, n)$-saturé maximal et $c = \max(\{\mathbf{sp}(n'), n' \leq n\})$. pour tout $k \leq c$, notons

$$S_k := \{(x_1, \ldots, x_k) \mid \mathbf{x} \in \psi(D)\} \qquad \text{et} \qquad M_k := \{f^{(k)}(\mathbf{x}) \mid \mathbf{x} \in \psi(D)\}.$$

Comme $D$ est le diagramme $(\mathbf{sp}, n)$-saturé maximal, l'ensemble $S_c$ est l'ensemble $(\mathbf{sp}, n)$-saturé contenant le plus d'états et $M_c$ est un intervalle. De plus pour tout $k \leq c$, $M_k$ est un intervalle. Pour tout $\mathbf{x}, \mathbf{y} \in \psi(D)$ on a $f^{(c)}(\mathbf{x}) \neq f^{(c)}(\mathbf{y})$, ainsi on a $|S_c| = |M_c|$. Comme $M_c$ est un intervalle, chaque nœud $(k, \ell)$ tel que $k \leq c$ a donc au plus un arc entrant.

Soit $k \leq c$, on a par définition de $\psi(D)$ :

$$V(k, \ell) = \{v \mid \mathbf{x} \in S_c,\ f^{(k)}(\mathbf{x}) = \ell \text{ et } f^{(k+1)}(\mathbf{x}) = \ell + v\}.$$

Comme $S_c$ est maximal, $V(k, \ell)$ est un intervalle. Ce qui montre la condition $i)$ de la propriété $\mathcal{T}_c$.

Supposons qu'il existe $k$ et $c$ tels que $k \leq c$, $V(k, \ell) = \emptyset$ et $V(k, \ell + 1) = \emptyset$. Nous venons de monter que $V(k, \ell)$ et $V(k, \ell + 1)$ sont des intervalles. De plus les nœuds en colonne $(k + 1)$ possèdent au plus un arc entrant et $M_k$ est un intervalle. Ainsi la condition $ii)$ de la propriété $\mathcal{T}_c$ est vérifiée. □

### Colonne $m$-complète

Soit $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ le diagramme complet et soit $D = (\mathcal{N}, A) \subseteq \mathcal{D}$ un diagramme. On dit que la colonne $k$ de $D$ est **complète** si elle contient autant d'arcs que celle du diagramme

complet, c'est-a-dire si $A(k) = \mathcal{A}(k)$. Dans un diagramme complet, on note $W(k,\ell)(= V(k,\ell))$ l'ensemble des valeurs des arcs sortants du nœud $(k,\ell)$ :

$$W(k,\ell) := \{v \mid \big((k-1,\ell),(k,\ell+v)\big) \in \mathcal{A}\}.$$

Comme précédemment, on définit $w_{\min}(k,\ell) = \min(W(k,\ell))$ et $w_{\max}(k,\ell) = \max(W(k,\ell))$. On note de plus $\ell_{\min} = \min(\{\ell \mid W(k,\ell) \neq \emptyset\})$, $\ell_{\max} = \min(\{\ell \mid W(k,\ell) \neq \emptyset\})$, $\ell'_{\min} = \min(\{\ell \mid W(k+1,\ell) \neq \emptyset\})$ et $\ell'_{\max} = \min(\{\ell \mid W(k+1,\ell) \neq \emptyset\})$.

Dans un diagramme complet, si $\ell_{\min} > 0$, alors $W(k,\ell_{\min}) = \{C_k\}$. Cela correspond au chemin dans lequel les clients sont concentrés dans les files $k'$ avec $k' \geq k$. La figure 3.14 illustre un diagramme complet dans lequel $W(3,0) = \emptyset$ et $W(3,1) = \{C_3\}$.

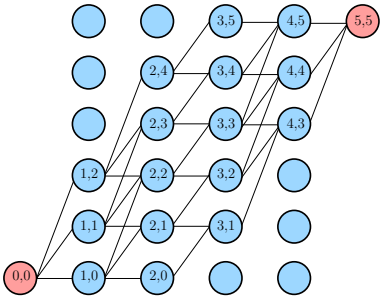



**FIGURE 3.14:** Diagramme complet pour $K = 5$, $M = 5$ et $\mathbf{C} = (2,2,1,2,2)$.

Remarquons que pour tout $D \subseteq \mathcal{D}$, on a pour chaque nœud $(k,\ell) \in \mathcal{N}$, $V(k,\ell) \subseteq W(k,\ell)$.

**Définition 11.** *La colonne $k$ d'un diagramme $D$ est dite $m$-**complète** si*

$$V(k,\ell) = \begin{cases} W(k,0) & \textit{si } \ell = 0 \\ [\max(\ell'_{\min} - \ell, m), \min(C_k, \ell'_{\max} - \ell)] & \textit{si } 0 < \ell \leq \min(\ell_{\max} - m, \ell'_{\max} - C_k) \textit{ et } m \leq C_k \\ \{C_k\} & \textit{si } \ell_{\max} - \min(m, C_k) < \ell \leq \ell'_{\max} - C_k \\ \emptyset & \textit{sinon.} \end{cases}$$

La figure 3.15 illustre cette définition pour $m \in \{0,1,\ldots,C_k\}$. On peut remarquer que toute colonne $0$-complète est complète (puisque elle correspond à celle du diagramme complet). Notons de plus que si une colonne $k$ est $C_k$-saturée alors pour tout $\ell \neq 0$ on a $V(k,\ell) = \{C_k\}$ ou $V(k,\ell) = \emptyset$.

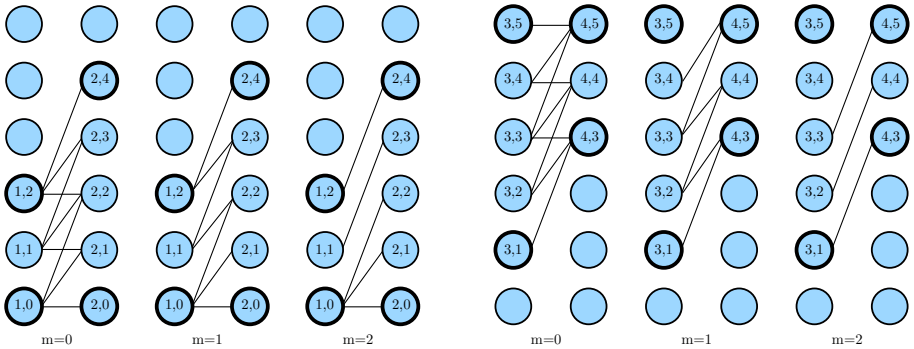



**FIGURE 3.15:** Exemples de colonnes $m$-complètes pour $m \in \{0,1,2\}$ et $C_k = 2$.

**Lemme 15.** *Soit $D$ un diagramme $\mathbf{sp}(n)$-saturé avec $c-1 = \max\{\mathbf{sp}(n'), n' \leq n\}$. Soit $i = \mathbf{sp}(n)$. Si la colonne $c$ est $m$-complète alors la colonne $c$ du diagramme $T_{i,c}(D)$ est $(m+1)$-complète.*

*Démonstration.* Soit $D$ un diagramme $\mathbf{sp}(n)$-saturé avec $c-1 = \max\{\mathbf{sp}(n'), n' \leq n\}$ et $i = \mathbf{sp}(n)$. On suppose que la colonne $c$ du diagramme $D$ est $m$-complète. Ainsi, dans le diagramme $D$ nous avons

$$V(c,\ell) = \begin{cases} W(c,0) & \text{si } \ell = 0 \\ [\max(\ell'_{\min} - \ell, m), \min(C_c, \ell'_{\max} - \ell)] & \text{si } 0 < \ell \leq \min(\ell_{\max} - m, \ell'_{\max} - C_c) \text{ et } m \leq C_c \\ \{C_c\} & \text{si } \ell_{\max} - \min(m, C_c) < \ell \leq \ell'_{\max} - C_c \\ \emptyset & \text{sinon.} \end{cases}$$

Soit $D' = T_{i,c}(D)$, on pose $V'(k,\ell)$ l'ensemble de toutes les valeurs des arcs ayant pour origine le nœud $(k-1,\ell)$ dans le diagramme $D'$.

Si $\ell = 0$ il y a deux cas possibles : $V(c,0) = \emptyset$ ou $V(c,0) \neq \emptyset$. Si $V(c,0) = \emptyset$ ceci implique que $\mathcal{S}tay = \emptyset$ et $V(c,1) \in \{\emptyset, \{C_c\}\}$ dans ce cas on a alors $V'(c,0) = \emptyset = V(c-1,0) = W(c-1,0)$. Si $V(c,0) \neq \emptyset$, puisque le diagramme $D$ est $\mathbf{sp}(n)$-saturé, l'ensemble de ses arcs dans $\mathcal{S}tay$ ($s=1$) en colonne $c$ à pour origine le nœud $(c-1,0)$. Ainsi dans un diagramme $\mathbf{sp}(n)$-saturé l'ensemble des arcs de valeurs $0$ des colonnes $k < c$ est contenu dans $\mathcal{S}tay$ et relit le nœud source $(0,0)$ au nœud $(c-1,0)$. Les arcs sortant de $(c-1,0)$ ne sont donc pas affecté par la transition $T_{i,c}(D)$ et on a $V'(c,0) = V(c,0) = W(c,0)$.

Pour $\ell > 0$, les arcs $\big((c-1,\ell),(c,\ell')\big)$ du diagramme $D$ appartiennent soit à l'ensemble $\mathcal{T}ransit$ soit à l'ensemble $\mathcal{F}ull$ ceci dépend de leurs valeurs. Si $m \geq C_c$ alors tous ses arcs sont dans $\mathcal{F}ull$ et ne seront pas affectés par la transition. On a on a $V'(c,0) = V(c,0)$, ainsi la colonne $c$ de $D'$ est aussi $(m+1)$-complète. Considérons maintenant $m < C_c$. Soit $I$ un sous ensemble d'entiers, on note $I_\uparrow = \{i+1 \mid i \in I\} \cap [0, C_c]$. On a

$$V'(c,\ell) = (V(c,\ell+1) \setminus \{C_c\})_\uparrow \cup (\{C_c\} \cap V(c,\ell)).$$

Supposons que $\ell + 1 \leq \min(\ell_{\max} - m, \ell'_{\max} - C_c)$. Alors $\ell \leq \max(\ell_{\max} - (m+1), \ell'_{\max} - C_c)$ et en utilisant le fait que $C_c \in V(c,\ell)$, alors $C_c - 1 \in V(c,\ell+1)$,

$$\begin{aligned} V'(c,\ell) &= [\max(\ell'_{\min} - \ell - 1, m), \min(C_c - 1, \ell'_{\max} - \ell - 1)]_\uparrow \cup (\{C_c\} \cap V(c,\ell)) \\ &= [\max(\ell'_{\min} - \ell, m+1), \min(C_c, \ell'_{\max} - \ell)], \end{aligned}$$

Maintenant supposons que $\ell_{\max} - m < \ell + 1 \leq \ell'_{\max} - C_c$. Ceci implique que $\ell_{\max} - (m+1) < \ell \leq \ell'_{\max} - C_c$. Nous avons $V(c,\ell+1) = \{C_c\}$ et $C_c \in V(c,\ell)$, ainsi $V'(c,\ell) = \{C_k\}$.

Pour finir, si $\ell + 1 < \max(\ell_{\max} - m, \ell'_{\max} - C_c)$, alors $V(c,\ell+1) = \emptyset$ et $V'(c,\ell) = \{C_c\}$. Si $C_c \in V(c,\ell)$, alors $\ell \leq \ell'_{\max} - C_c$. Dans le cas contraire, $V'(c,\ell) = \emptyset$. □

### 3.5.2 Preuve pour les diagrammes

**Théorème 8.** *Il existe une suite finie de transitions $T = T_{i_1,j_1,s_1} \circ \ldots \circ T_{i_N,j_N,s_N}$ tel que $\psi(T(\mathcal{D})) = 1$.*

*Démonstration.* Soit $\mathbf{sp}$ un chemin couplant associé au réseau. Nous allons procéder par récurrence sur les préfixes de $\mathbf{sp}$ en montrant qu'il existe une fonction $T_n$ qui transforme le diagramme complet en un diagramme $(\mathbf{sp}, n)$-saturé maximal, *i.e.* $T_n(\mathcal{D}) = \mathcal{D}_{(\mathbf{sp},n)}$.

*Initialisation :* Le diagramme complet $\mathcal{D}$ est le diagramme $(\mathbf{sp}, 1)$-saturé maximal, *i.e.* $\mathcal{D} = \mathcal{D}_{(\mathbf{sp},1)}$.
*Hérédité :* Soit $\mathcal{D}_{(\mathbf{sp},n)}$ le diagramme $(\mathbf{sp}, n)$-saturé maximal. On pose $i = \mathbf{sp}(n)$, $j = \mathbf{sp}(n+1)$ et $c - 1 = \max\{\mathbf{sp}(n') \mid n' < n\}$. Deux choses peuvent alors se produire,

- $j < c$ : on obtient $\mathcal{D}_{(\mathbf{sp},n+1)}$ à partir de $\mathcal{D}_{(\mathbf{sp},n)}$ en utilisant des transitions $T_{i',j'}$ avec $i', j' < c$ (lemme 13). Un exemple d'une telle suite de transitions est $T' = (T^{C_j}_{\mathbf{sp}(n-1),j} \circ \ldots \circ T^{C_2}_{1,2})^M$ ainsi $T_{n+1} = T' \circ T_n$.
- $j = c$ : La colonne $c$ du diagramme $\mathcal{D}_{(\mathbf{sp},n)}$ est complète (donc $0$-complète) car nous n'avons jusqu'à présent utilisé que des transitions $T_{i',j'}$ pour lesquelles $i', j' < c$.
  Le lemme 15, implique qu'en appliquant $C_c$ fois la transition $T_{i,c}$ on obtient un diagramme dans lequel la colonne $c$ est $C_k$-complète. Ce diagramme correspond donc au diagramme $\mathbf{sp}(n+1)$-saturé maximal. Ainsi $\mathcal{D}_{(\mathbf{sp},n+1)} = T_{i,c}(\mathcal{D}_{(\mathbf{sp},n)})^{C_c}$ et $T_{n+1} = T^{C_c}_{i,c} \circ T_n$.

En appliquant la récurrence, pour $n = L$, on obtient $\mathcal{D}_{(\mathbf{sp},L)}$ qui est un diagramme ne contenant qu'un seul chemin.

□

### 3.5.3 Preuve pour les diagrammes sans trou

Pour $n \in \{1, 2, 3\}$, on dit que la colonne $k$ d'un diagramme $D$ satisfait la propriété sans trou $k - \mathcal{GF}n$ si $\mathcal{GF}n$ est satisfaite pour colonne $k$ du diagramme.

**Lemme 16.** *Si un diagramme satisfait la propriété arbre $\mathcal{T}_c$ alors il satisfait aussi les propriétés sans trou $k - \mathcal{GF}1$, $k - \mathcal{GF}2$ et $k - \mathcal{GF}3$ pour tout $k \leq c$.*

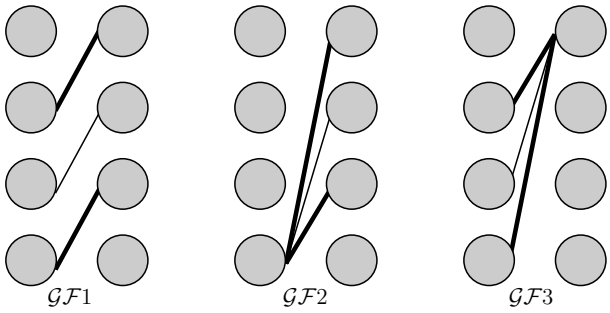



**FIGURE 3.16:** Conditions pour un diagramme sans trou.

*Démonstration.* Considérons $D = (\mathcal{N}, A)$ un diagramme satisfaisant $\mathcal{T}_c$ et $k \leq c$.

$(k-\mathcal{GF}1)$ : Supposons que $\big((k-1,\ell),(k,\ell+v)\big)$, $\big((k-1,\ell'),(k,\ell'+v)\big) \in A$ avec $0 \leq \ell < \ell' < M$. On a $v \in V(k,\ell) \cap V(k,\ell')$, ainsi $v_{\min}(k,\ell') \leq v \leq v_{\max}(k,\ell)$. De plus, $ii)$ et $iii)$ impliquent que pour tout $\ell''$ tel que $\ell < \ell'' < \ell'$, $v_{\max}(k,\ell) \leq v_{\min}(k,\ell'') \leq v_{\max}(k,\ell'') \leq v_{\min}(k,\ell')$. Ainsi, $v_{\max}(k,\ell) \leq v_{\min}(k,\ell') = v$ et $V(k,\ell'') = \{v\}$. Ce qui signifie, $\big((k-1,\ell''),(k,\ell''+v)\big) \in A$.

$(k - \mathcal{GF}2)$ : Supposons que $\big((k-1,\ell),(k,\ell+v)\big)$, $\big((k-1,\ell),(k,\ell+v')\big) \in A$, avec $0 \leq v < v' \leq C_k$. Ainsi $v, v' \in V(k,\ell)$. Puisque $V(k,\ell)$ est un intervalle, pour tout $v''$ tel que $v < v'' < v'$ on a $v'' \in V(k,\ell)$ ce qui implique $\big((k-1,\ell),(k,\ell+v'')\big) \in A$.

($k-\mathcal{GF}3$) : On procède par contradiction. Supposons que $\big((k-1,\ell-v'),(k,\ell)\big)$, $\big((k-1,\ell-v),(k,\ell)\big) \in A$ avec $0 \leq v < v' \leq C_k$. Alors $V(k,\ell-v')$ et $V(k,\ell-v)$ sont des intervalles qui contiennent respectivement $v'$ et $v$. Comme $\ell-v' < \ell-v$ on a $v' \leq v_{\max}(k,\ell-v') \leq v_{\min}(k,\ell-v) \leq v$. Ce qui est en contradiction avec $v' > v$. On en déduit alors que le nœud $(k,\ell)$ possède au plus un arc entrant. Ce qui satisfait $k-\mathcal{GF}3$. □

**Lemme 17.** *Toute colonne $k$ d'un diagramme qui est $m$-complète satisfait les propriétés sans trou $k-\mathcal{GF}1$, $k-\mathcal{GF}2$ et $k-\mathcal{GF}3$.*

*Démonstration.* Considérons une colonne $m$-complète et $V$ l'ensemble des valeurs des arcs de cette colonne. On a alors $V = \cup_{\ell=0}^{M} V(k,\ell)$ (voir la définition 11), ainsi $V$ est un intervalle ce qui satisfait $k-\mathcal{GF}1$. De plus toujours d'après la définition 11, quelque soit $\ell$, $V(k,\ell)$ est défini comme un intervalle ce qui satisfait $k-\mathcal{GF}2$.

($k-\mathcal{GF}3$) : Montrons maintenant $k-\mathcal{GF}3$ par récurrence sur $m$. Pour $m=0$, la colonne correspond à celle du diagramme complet. Comme un diagramme complet est aussi un diagramme sans trou, la colonne 0-complète possède donc la propriété $k-\mathcal{GF}3$. Supposons maintenant que la colonne soit $m$-complète et possède la la propriété $k-\mathcal{GF}3$. On sait par le lemme 15 que pour tout un diagramme $D$ $\mathbf{sp}(n)$-saturé ayant sa colonne $k$ $m$-complète et $i < k$, la colonne $k$ du diagramme $D' = T_{i,k}(D)$ est $(m+1)$-complète. Notons $V_a(k,\ell)$ l'ensemble des valeurs des arcs entrant du nœud $(k,\ell)$ appartenant au diagramme $D$ et ${V_a}'(k,\ell)$ le même ensemble pour $D'$. En appliquant la transitions $T_{i,k}$, les arcs $a = \big((k-1,m),(k,\ell)\big)$ se transforment en arcs $a' = \big((k-1,m'),(k,\ell)\big)$ tels que $m' = m$ si $v(a) = C_k$ ou $a \in \mathcal{S}tay$ et $m' = m-1$ sinon. Si $a \in \mathcal{S}tay$, puisque $D$ est $\mathbf{sp}(n)$-saturé alors $v(a) = \max V_a(k,\ell)$. Ainsi si $V_a(k,\ell)$ est un intervalle, $V_a(k,\ell)'$ l'est aussi. □

**Théorème 9.** *Il existe une suite finie de transitions $T = T^{\mathcal{GF}}_{i_1,j_1,s_1} \circ \ldots \circ T^{\mathcal{GF}}_{i_N,j_N,s_N}$ tel que $\psi(T(\mathcal{D})) = 1$.*

*Démonstration.* Les diagrammes possédant la propriété d'arbre $\mathcal{T}_c$ satisfont $k-\mathcal{GF}1$, $k-\mathcal{GF}2$ et $k-\mathcal{GF}3$ pour $k \in \{1,\ldots,c\}$. De plus les colonnes $k$ $m$-complètes d'un diagramme satisfont aussi $k-\mathcal{GF}1$, $k-\mathcal{GF}2$ et $k-\mathcal{GF}3$. Ainsi un diagramme possédant la propriété arbre $\mathcal{T}_c$ et ayant pour ses colonnes d'indices supérieurs à $c$ que des colonnes $m$-complètes est alors sans trou.

Dans preuve du théorème 8 on ne traite donc que des diagrammes qui sont aussi sans trou. En appliquant les mêmes transitions que dans la preuve du théorème 8 on montre alors qu'il existe une suite de transitions sans trou transformant le diagramme complet en un diagramme ne contenant plus qu'un seul chemin.

□

# Conclusion

Grâce à la théorie développée au chapitre 2, l'espace des états d'un réseau fermé de $K$ files d'attentes et $M$ clients a pu être représenté avec une complexité mémoire en $O(KM^2)$ au lieu de $O(M^{K-1})$. Ainsi, avec la définition d'un algorithme effectuant des mises à jour directement sur un diagramme on a élaboré l'algorithme `PSD` qui est efficace pour la simulation parfaite des réseaux fermés de files d'attente monoclasse.

Dans le but d'améliorer encore la complexité de la représentation par diagramme, on a introduit les diagrammes sans trou. Bien qu'ils soient moins précis que les diagrammes pour la représentation des ensembles d'états, les simulations ont montré que cette nouvelle

représentation avait une influence limitée sur le temps de couplage de l'algorithme de simulation parfaite PSF, tout en permettant d'améliorer le temps d'exécution.

À l'aide de simulations, on a comparé les temps moyens de couplage des trois algorithmes de simulation parfaite PSS, PSD et PSF. Les simulations ont montré que pour une "bonne numérotation" du réseau de files d'attente, les temps de couplages étaient assez proches. Néanmoins, on ne possède pas de résultats théoriques concernant les temps de couplages des algorithmes PSS, PSD et PSF. La dépendance de la numérotation des files pour PSD et PSF laisse penser que de tels résultats sont loin d'êtres évidents.

Dans le chapitre 8 on s'intéressera à l'implémentation des diagrammes et des algorithmes de ce chapitre. On présentera la *toolbox* Clones qui permet la simulation de réseaux fermés monoclasse avec des diagrammes avec et sans trou. Dans le chapitre suivant, on généralisera la technique développée dans ce chapitre pour les réseaux fermés de files d'attente multiclasses.

# 4 Simulation parfaite de réseaux fermés multiclasses

Nous généralisons maintenant les résultats du chapitre 3 traitant de la simulation parfaite des réseaux fermés monoclasses aux réseaux fermés multiclasses. On commencera par définir le modèle qui est bien plus complexe que celui du chapitre 3 notamment car à chaque file est associée une discipline de service. Les diagrammes correspondant aux états des réseaux multiclasses sont ceux des sommes vectorielles cumulées du chapitre 2. On présentera un nouvel algorithme de transition agissant sur ces digrammes. L'algorithme de simulation parfaite pour ce modèle utilisera comme au chapitre 3, une représentation par diagramme et un algorithme de transition. Enfin, on mènera une étude expérimentale du temps de couplage.

## 4.1 Présentation du problème et modélisation

### 4.1.1 Description

On considère dans ce chapitre des réseaux fermés de files $K$ d'attentes possédant $Z$ classes de clients et $\mathcal{C} := \{1, \ldots, Z\}$ désigne l'ensemble des classes. Pour simplifier le modèle, on considère que chaque file $k \in \mathcal{Q} := \{1, \ldots, K\}$ a une capacité de taille infinie et un seul serveur.

Chaque file $k$ possède une discipline de service qui lui ait propre. Les disciplines possibles doivent obéir aux règles décrites en section 4.1.3. Pour simplifier encore le modèle, on considère que le taux de service d'une file $\mu_k$ ne dépend pas de la classe de ses clients. La généralisation à un modèle dans lequel le taux de service est dépendant de la classe des clients s'obtient en ajoutant des transitions qui ne modifient pas l'état des classes ayant un plus faible taux de service.

L'ensemble des classes des clients autorisés en files $k$ est noté $\mathcal{C}(k) \subseteq \mathcal{C}$. Les clients ne peuvent pas changer de classe. Ainsi le nombre de clients dans chaque classe est constant et pour $z \in \mathcal{C}$, on note $M_z$ le nombre de clients de classe $z$ dans le réseau. Le vecteur $\mathbf{M} = (M_1, \ldots, M_Z)$ indique le nombre de clients dans chaque classe, il est appelé le **vecteur de population**. Le nombre total de clients dans un réseau est $M = \sum_{z=1}^{Z} M_z$.

L'ensemble des files visitées par les clients de classe $z$ est noté $\mathcal{Q}(z) \subseteq \mathcal{Q}$ et $\mathbf{P}^{(z)} = (p_{i,j}^{(z)})_{i,j \in \mathcal{Q}}$ est la matrice de routage pour les clients de classe $z$. Par convention si $i \notin \mathcal{Q}(z)$ alors $p_{i,i}^{(z)} = 1$ et $p_{i,j}^{(z)} = 0$ pour tout $j \in \mathcal{Q} \setminus \{i\}$. On considère que $\mathbf{P}^{(z)}$ est stochastique, ainsi pour tout $i, j \in \mathcal{Q}$, $p_{i,j}^{(z)} \geq 0$ et $\sum_{j=1}^{K} p_{i,j}^{(z)} = 1$. Soit $G_z = (\mathcal{Q}(z), R_z)$ le graphe orienté tel que $R_z = \{(i,j) \in \mathcal{Q}(z)^2$ tel que $p_{i,j}^{(z)}$ et $i \neq j\}$, puisque $\mathbf{P}^{(z)}$ est irréductible, $G_z$ est fortement connexe.

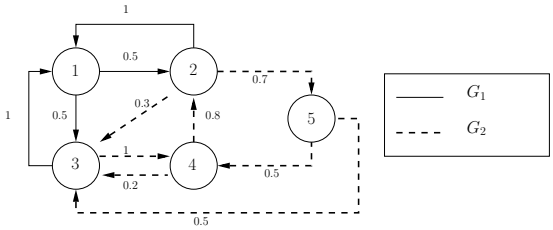



**FIGURE 4.1:** Réseau multiclasse décrit dans l'exemple 15.

### 4.1.2 Espace des états du réseau

Un état du réseau est représenté par une matrice $\mathbf{x} = (x_{z,k}) \in \mathbb{N}^{Z \times K}$ ou $x_{z,k}$ est le nombre de clients de classe $z$ en file $k$. Un état $\mathbf{x}$ doit satisfaire les contraintes suivantes :

1. pour tout $z \in \mathcal{C}$, $\sum_{k=1}^{K} x_{z,k} = M_z$,
2. pour tout $k \notin \mathcal{Q}(z)$, $x_{z,k} = 0$.

Pour $k \in \mathcal{Q}$ et $z \in \mathcal{C}$, on note

- $x_{z,*} = (x_{z,k}, \ldots, x_{z,K}) \in \mathbb{N}^K$ le vecteur ligne modélisant la répartition des clients de classe $z$ dans le réseau ;
- $x_{*,k} = (x_{1,k}, \ldots, x_{Z,k})^t \in \mathbb{N}^Z$ le vecteur ligne modélisant la répartition des clients dans la file $k$ selon leur classe ;
- $|x_{*,k}| = \sum_{z=1}^{Z} x_{z,k}$ le nombre total de clients en file $k$.

L'ensemble de tous les états possibles est donné par

$$\mathcal{S} = \left\{ \mathbf{x} \in \mathbb{N}^{Z \times K} \;\middle|\; \forall z \in \mathcal{C},\ \sum_{k=1}^{K} x_{z,k} = M_z \text{ et } \forall k \notin \mathcal{Q}(z),\ x_{z,k} = 0 \right\}.$$

Ainsi on en déduit

$$|\mathcal{S}| = \prod_{z=1}^{Z} \binom{M_z}{M_z + |K(z)| - 1}.$$

Si pour chaque classe $z$, $\mathcal{Q}(z) \ll M_z$ alors $|\mathcal{S}| = O(\prod_{z=1}^{Z} M_z^{|K(z)|}) = O(M_\times^K)$ avec $M_\times = \prod_{z=1}^{Z} M_z$.

**Exemple 15.** *On considère un réseau de 5 files possédant 2 classes de clients contenant 2 clients de classe 1 et 3 clients de classe 2 ($M_1 = 2$ et $M_2 = 3$). Le routage à l'intérieur de ce réseau est donné par les matrices suivantes :*

$$\mathbf{P}^{(1)} = \begin{pmatrix} 0 & 0.5 & 0.5 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad \textit{et} \quad \mathbf{P}^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.3 & 0 & 0.7 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0.8 & 0.2 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 & 0 \end{pmatrix}.$$

*Le nombre de clients total dans le réseau est $M = 5$, à partir des matrices de routage on déduit $\mathcal{Q}(1) = \{1, 2, 3\}$ et $\mathcal{Q}(2) = \{2, 3, 4, 5\}$. Il y a donc $|\mathcal{S}| = 120$ états possibles. En voici un exemple :*

$$\mathbf{x} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 1 \end{pmatrix} \in \mathcal{S}.$$

### 4.1.3 Discipline de service et transitions

Les transitions dépendent de la discipline de service de la file. La fonction $\tilde{t}_{i,j,z} : \mathcal{S} \to \mathcal{S}$ décrit la transition d'un client de classe $z$ de la file $i$ vers la file $j$

$$\tilde{t}_{i,j,z}(\mathbf{x}) = \mathbf{x} - \mathbb{1}_{\{x_{z,i}>0\}}(\mathbf{e}_{z,j} - \mathbf{e}_{z,j}),$$

avec $\mathbf{e}_{zk} \in \mathbb{N}^{Z\times K}$ la matrice qui a tous ses coefficients égaux à $0$ excepté $(\mathbf{e}_{zk})_{z,k} = 1$. Quand plusieurs classes de clients sont dans une file, alors la discipline détermine quel type de client servir en fonction de la configuration des clients dans la file. En cas de litige, on utilise une variable aléatoire pour déterminer la classe de client à servir. Pour chaque file $i \in \mathcal{Q}$, la fonction $f_i : \mathbb{N}^Z \times [0,1] \to \mathcal{C}(i) \cup \{0\}$ détermine la classe à servir dans la file $i$, la fonction retourne $0$ si aucune classe n'est choisie. On suppose que cette fonction a les propriétés suivantes

**Hypothèse 1.** *Pour $\mathbf{x} \in \mathcal{S}$ et $\theta \in [0,1]$ :*

1. *la discipline de service est markovienne et $f_i$ dépend seulement de $\theta$ et $x_{*,i}$.*
2. *$f_i(x_{*,i}, \theta) = 0$ si et seulement si $|x_{*,i}| = 0$.*
3. *Si $|x_{*,i}| > 0$ alors $f_i(x_{*,i}, \theta) \in \{z$ tel que $x_{z,i} > 0\}$.*

Voici deux exemples de discipline satisfaisant l'hypothèse 1.

- PRIORITY les clients sont choisis en fonction du numéro de leur classe par ordre croissant, ce qui se traduit par

$$f_i(x_{*,i}, \theta) = \min\{z \mid x_{z,i} > 0\}\mathbb{1}_{\{|x_{*,i}|>0\}}.$$

- LONGEST choisit la classe qui contient le plus de clients, ce qui se traduit par

$$f_i(x_{*,i}, \theta) \in \arg\max\{x_{i,z} \mid z \in Z(i)\} \ \vee \ 0.$$

Dans LONGEST on utilise $\theta$ pour déterminer la classe client à servir si plusieurs files ont la même taille. Il ne peut pas avoir de conflit pour PRIORITY ainsi cette discipline ne dépend pas de $\theta$.

Pour $i \in \mathcal{Q}$, on note $J = (j_1, \ldots, j_Z) \in \mathcal{Q}^Z$ le vecteur des destinations, il indique une destination pour chaque classe. La fonction $t_{i,J,\theta}(\mathbf{x}) : \mathcal{S} \to \mathcal{S}$ détermine (en fonction de la discipline en file $i$) la classe à servir et applique la transition selon la classe choisie. Plus précisément, pour $\mathbf{x} \in \mathcal{S}$ et $\theta \in [0,1[$ en posant $z = f_i(x_{*,i}, \theta)$ la classe choisie par la file $i$, on a :

$$t_{i,J,\theta}(\mathbf{x}) = \tilde{t}_{i,j_z,z}(\mathbf{x}).$$

Cette transition décrit le routage d'un client en file $i$ vers une destination $j$ appartenant à un ensemble de destinations décrites par le vecteur $J$. Si la transition ne dépend pas de la valeur $\theta$ (comme c'est le cas par exemple pour la discipline PRIORITY) alors on note $t_{i,J,\theta}(\mathbf{x}) = t_{i,J}(\mathbf{x})$.

**Exemple 16.** *Nous considérons à nouveau le réseau de l'exemple 15. Soit $\mathbf{x} \in \mathcal{S}$ un état tel que*

$$\mathbf{x} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 1 \end{pmatrix}.$$

*L'état $\mathbf{x}$ modélise une configuration du réseau dans laquelle la file $2$ contient $3$ clients, $1$ client de classe $1$ et $2$ de classe $2$ ($x_{*,2} = (1,2)$).*

*La fonction $t_{2,(1,5),0}$ ($\theta = 0$) décrit le routage suivant. Si la discipline la file 2 choisit un client de classe 1 alors il est servi puis dirigé en file 1. Dans le cas contraire, si elle choisit un client de classe 2 et ce dernier est dirigé en file 5. Notons $z = f_2(x_{*,2}, 0)$ la classe choisie par la discipline de la file 2. On s'intéresse à la valeur de $z$ selon cette discipline.*

- *Si la file 2 a pour discipline PRIORITY alors $z = 1$ et*

$$t_{2,(1,5)}(\mathbf{x}) = \tilde{t}_{2,1,1}(\mathbf{x}) = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 1 \end{pmatrix}.$$

- *Si la file 2 a pour discipline LONGEST alors $z = 2$ et*

$$t_{2,(1,5),0}(\mathbf{x}) = \tilde{t}_{2,5,2}(\mathbf{x}) = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 \end{pmatrix}.$$

La définition de $t_{i,J,\theta}$ avec l'utilisation du vecteur $J$ peut sembler étrange, alors que pour un état donné, on opère une transition que pour une seule classe. Cependant elle nous permet de définir une transition opérant sur un ensemble d'états. Pour tout $S \subseteq \mathcal{S}$, $\theta \in [0,1]$, $i \in \mathcal{Q}$ et $J = (j_1, \dots, j_Z) \in \mathcal{Q}^Z$, on définit ainsi

$$t_{i,J,\theta}(S) := \underset{\mathbf{x}\in S}{\cup} t_{i,J,\theta}(\mathbf{x}) \qquad \text{et} \qquad t_{i,J}(S) := \underset{\mathbf{x}\in S}{\cup} t_{i,J}(\mathbf{x}).$$

### 4.1.4 Chaîne de Markov et simulation parfaite

Soient $i \in \mathcal{Q}$ et $J = (j_1, \dots, j_Z) \in \mathcal{Q}^Z$, on note $(W_n)_{n\in\mathbb{N}}$ une suite i.i.d de variables aléatoires de distribution

$$\mathbb{P}(W_n = (i, J)) = \frac{\mu_i}{\sum_{k=1}^K \mu_k} \prod_{z=1}^{Z} p_{i,j_z}^{(z)},$$

Ainsi la probabilité que le service se produise dans la file $i \in \mathcal{Q}$ est égale à $\frac{\mu_i}{\sum_{k=1}^K \mu_k}$. De plus, en notant $W_n = (k, J)$, pour chaque classe $z \in \mathcal{C}$ on retrouve $\mathbb{P}(j_z = j \mid k = i) = p_{i,j_z}^{(z)}$ la probabilité pour qu'un client de classe $z$ qui vient d'être servi en file $i$ soit dirigé en file $j$.

Soit $(\Theta_n)_{n\in\mathbb{N}}$ une suite de variable i.i.d, uniformément distribuées sur $[0,1[$ et indépendantes de $(W_n)_{n\in\mathbb{N}}$. On pose $(U_n)_{n\in\mathbb{N}} = (W_n, \Theta_n)_{n\in\mathbb{N}}$.

On définit la chaîne de Markov pour notre modèle de réseau fermé multiclasse par $X_0 \in \mathcal{S}$ et

$$X_{n+1} = t_{U_n}(X_n).$$

Grâce à l'hypothèse 1 et aux matrices de routage fortement connexes, la chaîne $(X_n)_n$ est ergodique. Notre objectif est maintenant d'échantillonner la distribution stationnaire de cette chaîne à l'aide de l'algorithme de simulation parfaite. C'est ce que fait l'algorithme `PSSM` présenté ci-dessous (algorithme 13).

**Théorème 10** (Application directe de [51])**.** *L'algorithme `PSSM` termine avec probabilité 1 en un temps fini et renvoie $\mathbf{x} \in \mathcal{S}$ distribué selon la distribution stationnaire de $(X_n)_{n\in\mathbb{N}}$.*

Comme dans le chapitre 3, la principale difficulté de la preuve du théorème 10 est de montrer qu'il existe une suite de transitions transformant l'espace des états en un ensemble d'état ne contenant plus qu'un seul état ($t$ telle que $t(\mathcal{S}) = S$ avec $|S| = 1$). La preuve de ce résultat sera donné à la fin de chapitre, dans la section 4.2.4.

**Algorithme 13:** Algorithme PSSM

**Données :** $(U_{-n} = (i_{-n}, J_{-n}, \Theta_{-n}))_{n\in\mathbb{N}}$ une suite i.i.d de v.a.
**Résultat :** $\mathbf{x} \in \mathcal{S}$

**début**
  $n \leftarrow 1$;
  $t \leftarrow t_{U_{-1}}$;
  $S \leftarrow \mathcal{S}$;
  **tant que** $|t(S)| \neq 1$ **faire**
    $S \leftarrow \mathcal{S}$;
    $n \leftarrow 2n$;
    $t \leftarrow t_{U_{-1}} \circ \cdots \circ t_{U_{-n}}$;
  **renvoyer** $\mathbf{x}$, l'unique élément de $t(\mathcal{S})$

## 4.2 Simulation parfaite à l'aide des diagrammes

### 4.2.1 Espace des états et diagramme

Considérons $\mathcal{E} = \mathcal{G} = \mathbb{N}^Z$, $\forall k \in Q,\ \mathcal{E}_k = \mathbb{N}^{n_{k,1}} \times \ldots \times \mathbb{N}^{n_{k,Z}} \subseteq \mathbb{N}^Z$ avec $n_{k,z} = \mathbb{1}_{\{z\in\mathcal{C}(k)\}}$. La suite $\mathcal{F} = (f^{(k)})_{k\in\mathbb{N}}$ est définie telle que :

$$\begin{array}{rccc} f^{(k)} & : & \mathbb{N}^k & \to & \mathbb{N} \\ & & \mathbf{x} & \mapsto & (\sum_{i=1}^{k} x_{1,i}, \ldots, \sum_{i=1}^{k} x_{z,i}, \ldots, \sum_{i=1}^{k} x_{Z,i}) \end{array}$$

La suite $\mathcal{F} := (f^{(k)})_{k\in\mathbb{N}}$ est sans mémoire car elle s'inscrit dans le cadre du lemme 2 si on considère le monoïde $(\mathbb{N}^Z, +, (0, \ldots, 0))$. L'espace des $\mathcal{S}$ correspond à l'espace sans mémoire $\Omega(K, M)$ et on peut donc représenter tout sous-ensemble $S \subseteq \mathcal{S}$ par un diagramme (*cf.* chapitre 2).

La fonction $g$ qui transforme un état en un ensemble d'arcs est donnée par

$$\begin{array}{rccc} g & : & \mathcal{S} & \to & \mathcal{P}(\mathcal{A}) \\ & & \mathbf{x} & \mapsto & \bigcup_{k=1}^{K} \left\{ \left((k-1, \sum_{i=1}^{k-1} x_{1,i}, \ldots, \sum_{i=1}^{k-1} x_{Z,i}), (k, \sum_{i=1}^{k} x_{1,i}, \ldots, \sum_{i=1}^{k} x_{Z,i})\right)\right\}. \end{array}$$

Pour $D = (\mathcal{N}, A)$ un diagramme et un arc $a = \big((k-1, \mathbf{s}), (k, \mathbf{d})\big) \in A$, la définition de la valeur de l'arc $a$ est :

$$v(a) = (\sum_{i=1}^{k} x_{1,i} - \sum_{i=1}^{k-1} x_{1,i}, \ldots, \sum_{i=1}^{k} x_{Z,i} - \sum_{i=1}^{k-1} x_{Z,i}) = \mathbf{d} - \mathbf{s}.$$

Chaque diagramme a pour nœud source $(0, (0, \ldots, 0))$ et pour nœud destination $(K, (M_1, \ldots, M_Z))$. Le diagramme complet est donné par $\mathcal{D}(K, \mathbf{M}) = (\mathcal{N}, g(\mathcal{S}))$, on le notera aussi $\mathcal{D}$.

**Exemple 17.** *On considère $\mathcal{S}$ l'espace d'état de l'exemple 15 et l'état* $\mathbf{x} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 1 \end{pmatrix}$.
*Le diagramme $D_1 = (\mathcal{N}, g(\{\mathbf{x}\}))$ est donné en figure 4.2 et contient 5 arcs :*

$$\big((0,(0,0)),(1,(1,0))\big), \big((1,(1,0)),(2,(2,2))\big), \big((2,(2,2)),(3,(2,2))\big),$$
$$\big((3,(2,2)),(4,(2,2))\big), \big((4,(2,2)),(5,(2,3))\big).$$

*La valeur de l'arc* $a_2 = \big((1,(1,0)),(2,(2,2))\big)$ *est* $v(a_2) = (1,2) = x_{*,2}$. *Le diagramme complet est illustré en figure 4.3, il contient* $|g(\mathcal{S})| = 71$ *arcs.*

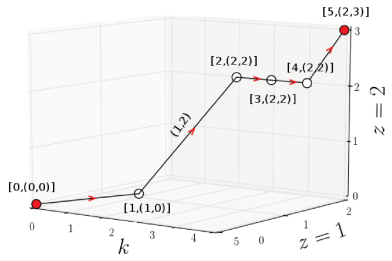



**FIGURE 4.2:** Diagramme $D_1$.

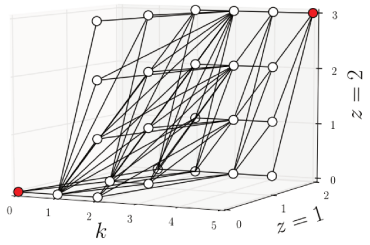



**FIGURE 4.3:** Diagramme complet.

**Lemme 18.** *Le nombre d'arcs d'un diagramme* $D = (\mathcal{N}, A)$ *est tel que*

$$|A| \leq 2\prod_{z=1}^{Z}(M_z+1) + (K-2)\prod_{z=1}^{Z}\frac{(M_z+1)(M_z+2)}{2}.$$

*Démonstration.* Le nombre d'états d'un réseau à $K$ files est maximal quand chaque file du réseau peut être visitée par tous les types de clients. Notons $\mathcal{D}$ le diagramme complet d'un tel réseau. Les colonnes $1$ et $K$ du diagramme complet contiennent $|\mathcal{A}(1)| = |\mathcal{A}(K)| = \prod_{z=1}^{Z} M_z + 1$ arcs. Pour les colonnes $k$ tel que $1 < k < K$ on a alors $|A(k)| = \prod_{z=1}^{Z} \frac{(M_z+1)(M_z+2)}{2}$. □

On en déduit que la complexité de la représentation par diagrammes est en $|\mathcal{A}| = O(KM_\times^2)$.

### 4.2.2 Algorithme de transition

Comme dans le chapitre 3, on définit la fonction de transition sur les diagrammes à l'aide des fonctions de transformation, d'un digramme vers un ensemble d'état ($\psi$) et réciproquement ($\phi$). Soient $(i,J) \in \mathcal{Q}^{Z+1}$, $\theta \in [0,1[$, on pose

$$T_{i,J,\theta} = \phi \circ t_{i,J,\theta} \circ \psi.$$

Le lemme suivant est l'analogue au lemme 9 du chapitre 3. Il permet l'application de l'algorithme de simulation parfaite.

**Lemme 19.** *Soit* $S \subseteq \mathcal{S}$ *un ensemble d'états et* $D$ *un diagramme. Alors pour tout* $(i,J) \in \mathcal{Q}^{Z+1}$ *et* $\theta \in [0,1[$,

1. *si* $S \subseteq \psi(D)$ *alors* $t_{i,J,\theta}(S) \subseteq \psi(T_{i,J,\theta}(D))$,
2. *si* $|\psi(D)| = 1$ *alors* $|\psi(T_{i,J,\theta}(D))| = 1$.

Comme dans le chapitre 3, on a recours à un algorithme qui permet d'effectuer la transition $T_{i,J,\theta}$ directement sur le diagramme. Expliquons le fonctionnement de cet algorithme de transition (*cf.* algorithme 14).

Dans la première section, nous avons défini :

$$t_{i,J,\theta}(\mathbf{x}) = \tilde{t}_{i,j_z,z}(\mathbf{x}) \qquad \text{avec} \qquad z = f_i(x_{*,i},\theta).$$

Chaque état $\mathbf{x} \in S$ peut être identifié à un chemin dans le diagramme $D = (\mathcal{N}, g(S))$ et réciproquement. Dans l'algorithme 14, on commence par grouper les ensembles d'arcs en fonction des ensembles d'états qu'ils encodent. On effectue ensuite les changements directement sur les arcs du diagramme selon les ensembles d'appartenance. On commence par traiter les arcs de la colonne $i$ en déterminant pour chaque arc la classe qui doit être servie. On sépare alors les arcs de la colonne $i$ en sous-ensembles disjoints. Puis pour chaque sous-ensemble d'arcs en colonne $i$, on détermine les arcs des autres colonnes étant liés au sous-ensemble. Pour finir, on applique la modification pour chaque arc du diagramme selon son sous-ensemble d'appartenance et le numéro de sa colonne.

**En colonne $i$** Pour effectuer la transition $t_{i,J,\theta}$ sur un état, on se sert de la fonction $f_i$ qui détermine la classe du client à choisir en fonction de la configuration de l'état en file $i$. Pour un diagramme, la configuration de l'état en file $i$ correspond à la valeur d'un arc en colonne $i$ ($v(a) = x_{*,i}$). On définit pour les diagrammes la fonction $F_i$ (l'analogue à $f_i$ pour les arcs) qui prend pour entrée la valeur d'un arc et retourne la classe à traiter. Ainsi pour $a \in A(i)$ et $\theta \in [0, 1[$, on pose $F_i(a, \theta) = f_i(v(a), \theta)$. Il y a au plus $\mathcal{C}(i) + 1$ valeurs possibles pour $F_i(a, \theta)$. Les arcs en colonnes $i$ peuvent ainsi être séparés en $\mathcal{C}(i) + 1$ sous ensembles disjoints. Pour tout $z \in \mathcal{C}(i) \cup \{0\}$ on pose

$$P[z] = \{a \in A(i) \mid F_i(a, \theta) = z\}.$$

L'ensemble $P[0]$ correspond aux arcs tels que $v(a) = (0, \dots, 0)$, c'est-à-dire ceux représentant des états dans lesquels la file $i$ ne contient aucun client. On a ainsi $A(i) = \cup_{z \in \mathcal{C}(i) \cup \{0\}} P[z]$ avec pour tout $z \neq z'$, $P[z] \cap P[z'] = \emptyset$.

**Dans les autres colonnes** Dans le chapitre 2, on a défini $path(b)$ l'ensemble des arcs pouvant être reliés à l'arc $b \in A$ et $\mathcal{P}ath(B)$ l'ensemble des arcs pouvant être reliés à un ensemble d'arcs $B \subseteq A$. L'exemple-ci dessous illustre ses définitions.

**Exemple 18.** *Soit $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ le diagramme complet associé au réseau de l'exemple 15 et $a = \big((2,(2,0)),(3,(2,0))\big) \in \mathcal{A}(3)$, $b = \big((2,(1,0)),(3,(2,0))\big) \in \mathcal{A}(3)$ deux arcs en colonne 3. Pour $B = \{a, b\} \subseteq \mathcal{A}(3)$. Les ensembles $path(b)$ et $path(B)$ sont illustrés dans les figures 4.4 et 4.5.*

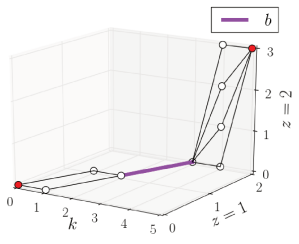



**FIGURE 4.4:** $path(b)$.

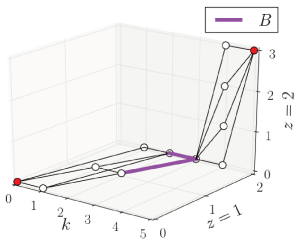



**FIGURE 4.5:** $\mathcal{P}ath(B)$.

Pour chaque $z \in \mathcal{C}(i) \cup \{0\}$, on note

$$\mathcal{S}erve[z] = \mathcal{P}ath(P[z]).$$

Remarquons que pour $z_1 \neq z_2$, les ensembles d'arcs $\mathcal{S}erve[z_1]$ et $\mathcal{S}erve[z_2]$ ne sont pas forcément disjoints. Les arcs contenus dans $\mathcal{S}erve[0]$ correspondent aux arcs des chemins pour lesquels la valeur des arcs en colonnes $i$ est égale à $(0, \dots, 0)$. Pour tout $z \in Z(i)$, $\mathcal{S}erve[z]$ est l'ensemble d'arcs qui correspond à l'ensemble d'états pour lesquels un client de classe $z$ est choisi pour le service. Le service s'effectue alors de la file $i$ vers la file $j = J_z$. Ainsi pour calculer la transition $T_{i,J,\theta}$, $\mathcal{S}erve[z]$ sera transformé en $\mathcal{S}erve[z]'$.

**Modifier les arcs** On suppose que $i < j$ (le cas $i > j$ utilise les mêmes idées et est détaillé dans l'algorithme 14). Chaque arc $a = \big((k-1, \mathbf{s}), (k, \mathbf{d})\big) \in \mathcal{S}erve[k]$ avec $z > 0$ est transformé en arc $b \in \mathcal{S}erve[k]'$ de la manière suivante :

- Si $k < i$ ou $k > j$ alors $a$ n'est pas affecté et $b = a$.
- Si $k = i$ alors $a$ correspond à une configuration de la file $i$ dans laquelle un client de classe $z$ est servi. Ainsi $b = \big((k-1, \mathbf{s}), (k, \mathbf{d} - \mathbf{e}_z)\big)$.
- Si $i < k < j$ alors $a$ correspond à une configuration de la file qui n'est pas affectée par le service. Mais la destination des arcs en colonne $k-1$ ont été affectés par la transition. Ainsi pour préserver les chemins on doit transformer l'origine $\mathbf{s}$ en $\mathbf{s} - \mathbf{e}_z$ pour maintenir la connexion. La destination de l'arc doit elle aussi changer pour conserver la valeur de l'arc. On obtient alors $b = \big((k-1, \mathbf{s} - \mathbf{e}_z), (k, \mathbf{d} - \mathbf{e}_z)\big)$.
- Si $k = j$, $a$ correspond à une configuration de la file $j$ dans laquelle un client de classe $z$ arrive. Mais l'origine de cet arc doit être affectée par la transition afin de maintenir la connexion. Ainsi $\mathbf{s}$ est transformé en $\mathbf{s} - \mathbf{e}_z$, nous avons donc $b = \big((k-1, \mathbf{s} - \mathbf{e}_z), (k, \mathbf{d})\big)$. Ce qui conserve le nœud de destination et ajoute un client de classe $z$.

Soit $\mathbf{x} \in \mathcal{S}$ tel que $|x_{*,i}| > 0$ et $f_i(x_{*,i}, \theta) = z$. On note $\mathbf{y} = t_{i,J,\theta}(\mathbf{x})$ alors on peut vérifier que pour chaque $c$ défini dans l'algorithme 14 on a $v(c) = y_{*,i}$.

**Transition** On pose $\mathcal{S}erve[0]' = \mathcal{S}erve[0]$. Soit $D = (\mathcal{N}, A)$, on a $A = \cup_{z \in \mathcal{C}(i) \cup \{0\}} \mathcal{S}erve[z]$ et on définit la transition sur le diagramme telle que $T_{i,J,\theta}(D) = (\mathcal{N}, A')$ avec

$$A' = \bigcup_{z \in \mathcal{C}(i) \cup \{0\}} \mathcal{S}erve[z]'.$$

Cette construction assure que pour tout $\mathbf{x} \in \psi(D)$, on a $t_{i,J,s}(\mathbf{x}) \in \psi(T_{i,J,s}(D))$.

**Algorithme 14:** Algorithm $T_{i,J,\theta}$

**Données** : $D = (\mathcal{N}, A)$, $i \in \mathcal{Q}$, $J \in \mathcal{Q}^Z$, $\theta \in [0, 1]$
**Résultat** : $T_{i,J,\theta}(D)$

**début**
  **pour** $z = 0$ *to* $Z$ **faire** $P[z] \leftarrow \{a \in A_i \mid F_i(a, \theta) = z\}$;
  $\mathcal{S}erve[0]' \leftarrow \mathcal{P}ath(P[0])$;
  **pour** $z = 1$ *to* $Z$ **faire**
    $\mathcal{S}erve[z] \leftarrow \mathcal{P}ath(P[z])$;
    $\mathcal{S}erve[z]' \leftarrow \emptyset$;
    $j \leftarrow J_z$;
    **si** $i < j$ **alors**
      **pour chaque** $b = \big((k-1, \mathbf{s}), (k, \mathbf{d})\big) \in \mathcal{S}erve[z]$ **faire**
        $c = \big((k-1, \mathbf{s} - \mathbb{1}_{\{i<k\leq j\}}\mathbf{e}_z), (k, \mathbf{d} - \mathbb{1}_{\{i\leq k<j\}}\mathbf{e}_z)\big)$;
        $\mathcal{S}erve[z]' \leftarrow \mathcal{S}erve[z]' \cup \{c\}$;
    **si** $i > j$ **alors**
      **pour chaque** $b = \big((k-1, \mathbf{s}), (k, \mathbf{d})\big) \in \mathcal{S}erve[z]$ **faire**
        $c = \big((k-1, \mathbf{s} + \mathbb{1}_{\{j<k\leq i\}}\mathbf{e}_z), (k, \mathbf{d} + \mathbb{1}_{\{j\leq k<i\}}\mathbf{e}_z)\big)$;
        $\mathcal{S}erve[z]' \leftarrow \mathcal{S}erve[z]' \cup \{c\}$;
  $A' \leftarrow \bigcup_{z=0}^{Z} \mathcal{S}erve[z]'$;
  **renvoyer** $(\mathcal{N}, A')$

Supposons que la complexité du calcul de $F_i(a, \theta)$ soit égal à une constante notée $C$. Alors la complexité de l'algorithme 14 est $C|A(i)| + Z|A| = O((C + KZ)M_\times^2)$. Par exemple pour les disciplines PRIORITY et LONGEST on a $C = O(Z)$ ainsi la complexité de l'algorithme de transition est en $O(KZM^{2Z})$.

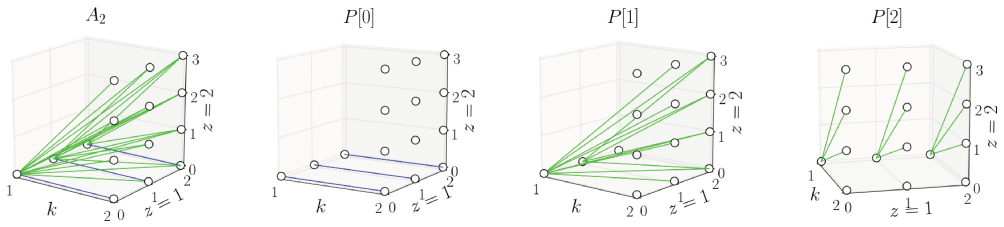



**FIGURE 4.6:** Répartition des arcs en colonne 2 avec la discipline PRIORITY.

**Exemple 19.** *On considère le diagramme complet de la figure 4.3 et on veut appliquer la transition $T_{2,(1,5)}$. On suppose de plus que la file 2 a une discipline PRIORITY (les clients de classe 1 sont prioritaire sur les clients de classe 2). La figure 4.6 illustre la répartition des arcs en colonne 2 selon la classe ($P[0]$, $P[1]$ et $P[2]$).*

- $P[0] = \{a \in A(2) \mid v(a) = (0,0)\}$ *et* $\mathcal{S}erve[0] = \mathcal{P}ath(P[0])$
- $P[1] = \{a \in A(2) \mid v(a) = (v_1, v_2),\ v_1 > 0\}$ *et* $\mathcal{S}erve[1] = \mathcal{P}ath(P[1])$
- $P[2] = \{a \in A(2) \mid v(a) = (0, v_2),\ v_2 > 0\}$ *et* $\mathcal{S}erve[2] = \mathcal{P}ath(P[2])$

*Les ensembles $P[0]$, $P[1]$ et $P[2]$ sont disjoints ce qui n'est pas le cas des ensembles $\mathcal{S}erve[0]$, $\mathcal{S}erve[1]$ et $\mathcal{S}erve[2]$. Par exemple, l'arc $b = \big((4,2,3),(5,2,3)\big) \in \mathcal{S}erve[1] \cap \mathcal{S}erve[1]$ (voir les figures 4.7 et 4.8). Pour tout $z \in \{0,1,2\}$ on peut alors calculer $\mathcal{S}erve[z]'$ à partir de $\mathcal{S}erve[z]$ selon l'algorithme 14. Les arcs des colonnes 1 et 2 en $\mathcal{S}erve[1]$ sont modifiés tandis que pour $\mathcal{S}erve[2]$ ce sont ceux des colonnes 2 à 5 qui le sont. On obtient ainsi $T_{i,J}(D) = (\mathcal{N}, A')$, avec $A' = \mathcal{S}erve[0] \cup \mathcal{S}erve[1]' \cup \mathcal{S}erve[2]'$.*

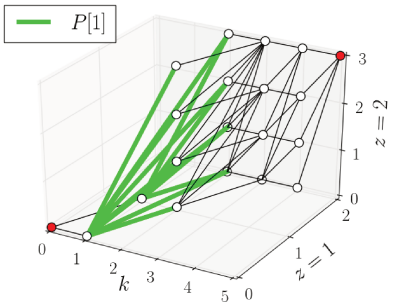



**FIGURE 4.7:** $\mathcal{S}erve[1] = \mathcal{P}ath(P[1])$.

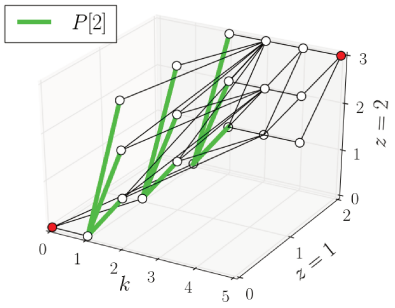



**FIGURE 4.8:** $\mathcal{S}erve[2] = \mathcal{P}ath(P[2])$.

### 4.2.3 Simulation parfaite

**Théorème 11** (Simulation parfaite avec diagrammes). *L'algorithme PSDM termine avec probabilité 1 en un temps d'espérance finie et retourne $\mathbf{x} \in \mathcal{S}$ distribué selon la distribution stationnaire de $(X_n)_{n\in\mathbb{N}}$.*

Comme dans le chapitre 3, on suppose qu'il existe une suite finie de transitions transformant le diagramme complet $\mathcal{D}$ en un diagramme ne possédant plus qu'un seul chemin. La démonstration du théorème est similaire à celle du théorème 6 (chapitre 3) en considérant les résultats du lemme 19 à la place de ceux du lemme 9. La preuve d'existence d'une suite de transitions couplante est donnée en section 4.2.4.

**Algorithme 15:** Algorithme PSDM

**Données :** $(U_{-n} = (i_{-n}, J_{-n}), \Theta_{-n})_{n\in\mathbb{N}}$ une suite i.i.d de v.a.
**Résultat :** $\mathbf{x} \in \mathcal{S}$

**début**
  $n \leftarrow 1$;
  $T \leftarrow T_{U_{-1}}$;
  **tant que** $|\psi(T(\mathcal{D}))| \neq 1$ **faire**
    $n \leftarrow 2n$;
    $T \leftarrow T_{U_{-1}} \circ \cdots \circ T_{U_{-n}}$;
  **renvoyer** $\mathbf{x}$, *l'unique élément de* $\psi(T(\mathcal{D}))$

### 4.2.4 Temps de couplage

On s'intéresse ici au temps de couplage des algorithmes PSSM et PSDM, c'est-à-dire au nombre de transitions $n$ effectuées quand un algorithme de simulation parfaite renvoie l'échantillon.

#### Réseau de l'exemple 15

On considère le réseau de l'exemple 15 avec pour vecteur population $\mathbf{M} = (m, m)$. La figure 4.9 illustre les temps de couplage des algorithmes PSSM et PSDM pour $m \in \{1, 2, \ldots, 20\}$. Pour chaque $m$ on effectue 100 simulations. Pour chaque simulation, on utilise deux fois la suite aléatoire $(U_{-n})$ (une fois par algorithme). Pour $m > 15$, le nombre $|\mathcal{S}|$ est trop grand pour permettre l'exécution de l'algorithme PSSM. Pour $m \leq 15$ on constate que les temps de couplage des deux algorithmes sont très proches.

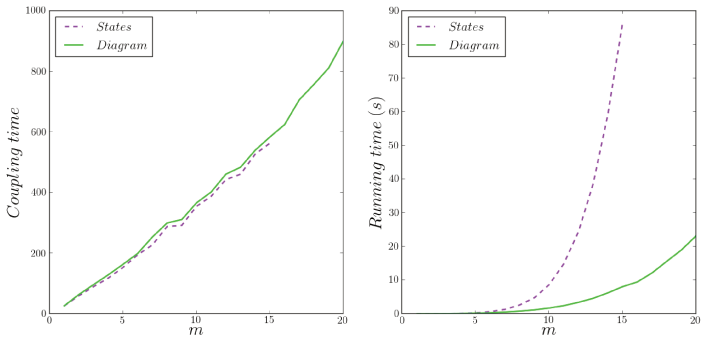



**FIGURE 4.9:** Comparaison des temps de couplage moyens (exemple 15).

#### Réseau en anneau bidirectionnel

On considère un réseau bidirectionnel à $K$ files, 2 classes ayant pour matrice de transitions $P^1_{i,i \mod (K)+1} = 0.9$, $P^1_{i \mod (K)+1,i} = 0.1$, $P^2_{i,i \mod (K)+1} = 0.1$ et $P^2_{i \mod (K)+1,i} = 0.9$. Il y a 5 clients dans chaque classe ($\mathbf{M} = (5, 5)$). On réalise l'expérimentation en faisant cette fois ci varier le nombre de files, $K \in \{2, 3, \ldots, 20\}$. Pour le peu de $m$ que nous pouvons tester ($m \leq 8$) du fait de la limitation induite par PSSM, on constate que les temps de couplage sont assez similaires (figures de droite).

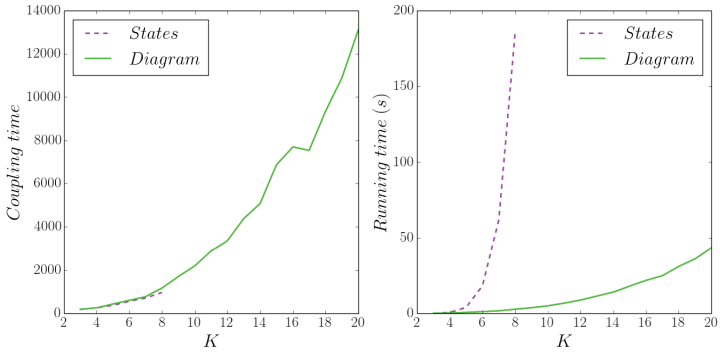



**FIGURE 4.10:** Comparaison des temps de couplage (réseau en anneau bidirectionnel).

### 4.2.5 Preuve d'existence d'une suite couplante de transitions

Dans cette section on montre qu'il existe une suite de transitions transformant l'espace des états en un ensemble d'états ne contenant plus qu'un seul état et que cette suite est aussi une suite couplante pour le diagramme.

Commençons par rappeler l'hypothèse 1 : Pour $\mathbf{x} \in \mathcal{S}$ et $\theta \in [0,1]$ :

1. la discipline de service est markovienne et $f_i$ dépend seulement de $\theta$ et $x_{*,i}$.
2. $f_i(x_{*,i}, \theta) = 0$ si et seulement si $|x_{*,i}| = 0$.
3. Si $|x_{*,i}| > 0$ alors $f_i(x_{*,i}, \theta) \in \{z \mid x_{z,i} > 0\}$.

#### Espace des états

**Théorème 12.** *Si chaque file satisfait l'hypothèse 1 alors il existe une suite finie de transitions $t$ telle que $|t(\mathcal{S})| = 1$.*

Dans cette démonstration on note $J = (J[1], J[2], \ldots, J[Z])$.

**Lemme 20.** *Soient $(i, J) \in \mathcal{Q}^{Z+1}$, $M$ le nombre total de clients dans le réseau et $(\theta_1, \ldots, \theta_{|M|}) \in [0,1[^M$, alors $\mathbf{y} = t_{i,J,\theta_M} \circ \ldots \circ t_{i,J,\theta_1}(\mathbf{x})$ satisfait les propriétés suivantes :*

1. *$|y_{*,i}| = 0$ (la file de départ $i$ est vide),*
2. *Pour tout $z \in \mathcal{C}$, $y_{z,J[z]} = x_{z,J[z]} + x_{z,i}$,*
3. *Pour tout $k \in \mathcal{Q} \setminus \{i, J[1], \ldots, J[Z]\}$ on a $y_{z,k} = x_{z,k}$.*

*Démonstration.* À chaque fois que la transition $t_{i,J,\theta}$ est appliquée, et ceci indépendamment de la valeur de $\theta$, le nombre de clients en file $i$ décroît de 1 s'il n'est pas nul. Comme $|x_i| \leq M$, en effectuant $M$ transitions on obtient $|y_i| = 0$. Chaque client de classe $z$ présent en file $i$ est dirigé en file $J[z]$ après $M$ transitions. □

Comme $t_{i,J,\theta_M} \circ \ldots \circ t_{i,J,\theta_1}$ ne dépend pas de $(\theta_1, \ldots, \theta_{|M|})$, on note $t^M_{i,J} := t_{i,J,\theta_M} \circ \ldots \circ t_{i,J,\theta_1}$. On se concentre maintenant sur ce qui se passe pour une classe $z$. Comme le réseau $G_z$ est fortement connexe, il existe un chemin $w = (w_1, w_2, \ldots, w_{N_z}) \in \mathcal{Q}(z)^{N_z}$ de taille $N_z$ et tel que $\{p^{(z)}_{w_n, w_{n+1}} > 0\}$ qui parcourt toutes les files de l'ensemble $\mathcal{Q}(z)$.

**Proposition 1.** *Soit $\mathbf{x} \in \mathcal{S}$ et $J_1, \ldots, J_{N_z} \in \mathcal{Q}^{N_z Z}$ tels que $\forall n \leq N_z - 1$, $J_n[z] = w_{n+1}$. Alors $\mathbf{y} = t^M_{w_{N_z-1}, J_{N_z-1}} \circ \cdots \circ t^M_{w_1, J_1}(\mathbf{x})$ satisfait $y_{z,w_{N_z}} = M_z$ et $y_{z,k} = 0$ pour tout $k \neq w_{N_z}$.*

*Démonstration.* On prouve ce résultat par récurrence. Posons $\mathbf{y}^{(n)} = t^{|M|}_{w_n,J_n} \circ \ldots \circ t^{|M|}_{w_1,J_1}(\mathbf{x})$. On va montrer que pour tout $n \leq N_z$,

$$y^{(n)}_{z,k} = \begin{cases} 0 & \text{si } k \in \{w_1, w_2, \ldots, w_n\} \setminus \{w_{n+1}\} \\ \sum\limits_{k \in \{w_1, w_2, \ldots, w_{n+1}\}} x_{z,k} & \text{si } k = w_{n+1} \\ x_{z,k} & \text{dans les autres cas.} \end{cases}$$

Le cas $n = 0$ est évident car $\mathbf{y}^{(0)} = \mathbf{x}$. Supposons que $\mathbf{y}^{(n-1)}$ satisfasse les propriétés énoncées ci-dessus. En appliquant le lemme 20 à $\mathbf{y}^{(n-1)}$ et $(w_n, J_n)$, on obtient

$$y^{(n)}_{z,w_{n+1}} = y^{(n-1)}_{z,w_n} + y^{(n-1)}_{z,w_{n+1}} = \sum_{i=1}^{n} x_{z,w_i} + x_{z,w_{n+1}} = \sum_{i=1}^{n+1} x_{z,w_i},$$

et que $y^{(n)}_{z,w_n} = y^{(n-1)}_{z,w_n} = 0$. Les autres files ne sont pas affectées en ce qui concerne leurs classe $z$, ainsi $\mathbf{y}^{(n)}$ satisfait les propriétés attendues. Pour conclure, remarquons que $\mathbf{y} = \mathbf{y}^{(N_z-1)}$ avec $\{w_i \mid i \in \{1, \ldots, N_z\}\} = \mathcal{Q}(z)$. □

Montrons maintenant le théorème 12.

*Démonstration.* Pour chaque $z \in \mathcal{C}$, on pose $t_z = t^M_{w_{N_z-1},J_{N_z-1}} \circ \cdots \circ t^M_{w_1,J_1}$ et $t = t_Z \circ \ldots \circ t_1$. La proposition 1 implique que chaque application de la fonction $t_z$ concentre tous les clients de la classe $z$ dans une même file. Le lemme 20 implique qu'une fois dans une même file après avoir appliqué $M$ fois une même transition les clients d'une même classe dans une même file restent groupés. Ainsi $t$ est bien une transition couplante. □

### Diagramme

**Théorème 13.** *Si chaque file satisfait l'hypothèse 1 alors il existe une suite finie de transitions $T$ telle que $|\psi(T(\mathcal{D}))| = 1$.*

*Démonstration.* La preuve est similaire à celle du théorème 12 en remplaçant $t_{i,J}$ par $T_{i,J}$ dans la suite de transitions $t$ que nous notons à présent $T$. Nous allons montrer que $T$ est une suite de transition couplante pour tout diagramme. L'idée est d'utiliser un équivalent au lemme 20 mais en un peu moins précis. Nous nous concentrons sur les diagrammes pour lesquels la classe $z$ a tout ses clients concentrés dans une seule file. On dit qu'un diagramme $D = (\mathcal{N}, A)$ satisfait $E_{z,k}$ si $\forall a \in A(k)$, $v(a)_z = 0$ (aucun client de classe $z$ en file $k$) et que $D$ satisfait $F_{z,k}$ si $\forall a \in A(k)$, $v(a)_z = M_z$ (tous les clients de classe $z$ sont en file $k$).

**Lemme 21.** *Soient $(i, J) \in \mathcal{Q}^{Z+1}$ et $(\theta_1, \ldots, \theta_M) \in [0, 1[^M$, on pose $D' = T_{i,J,\theta_M} \circ \cdots \circ T_{i,J,\theta_1}(D)$. Alors*

1. *$D'$ satisfait $E_{z,i}$ pour tout $z \in \mathcal{C}$ (*i.e. *la file $i$ est vide) ;*
2. *Si $D$ satisfait $E_{z,j}$ et $j \neq J[z]$ alors $D'$ satisfait aussi $E_{z,j}$ ;*
3. *Si $D$ satisfait $F_{z,i}$ alors $D'$ satisfait $F_{z,J[z]}$ ;*
4. *Si $D$ satisfait $F_{z,j}$, $j \neq i$ alors $D'$ satisfait aussi $F_{z,j}$.*

*Démonstration.* Pour commencer, considérons la colonne $i$. Chaque fois qu'une transition $T_{i,J,\theta_\ell}$ est appliquée, pour tout $\forall a \in A(i)$ le nombre total de clients représenté par l'arc $a$ $|v(a)| = \sum_{z=1}^{Z} v(0)$ décroit de 1 s'il est positif. Ainsi, après avoir appliqué $M$ fois cette transition sur un diagramme on a, $v(0) = (0, \ldots, 0)$ et ceux pour tout $a \in A(i)$.

Maintenant, considérons la classe $z$ et la file $j \neq J[z]$ telle que le diagramme $D$ satisfasse $E_{z,j}$. Par construction, pour tout $a \in A(j)$, $v(a)_z$ n'est pas affectée par la transition $T_{i,J}$ et ainsi $E_{z,j}$ est alors satisfait par $D'$.Si $D$ satisfait $F_{z,j}$, alors il satisfait aussi $E_{z,k}$ pour toute file $k \neq j$. □

Pour chaque classe $z$, on pose la suite de transition $T_z$ équivalente à $t_z$. Soit $D^{(n)} = T^M_{w_n,J_n} \circ \ldots \circ T^M_{w_1,J_1}(\mathbf{x})$. Le lemme 21 implique que $D^{(n)}$ satisfait $F_{z,w_n}$ et $E_{z,k}$ pour tout $k \in \{w_1, \ldots, w_{n-1}\} \setminus \{w_{n+1}\}$. Ainsi $D^{(N_z-1)}$ satisfait $E_{z,k}$ pour tout $k$ différent de $w_{N_z}$ et pour tout $a \in A(N_z)$, on a $v(a)_z = M_z$ et $D^{(N_z-1)}$ satisfait $F_{z,w_{N_z}}$.

En appliquant le lemme 21 on obtient que $T(D)$ satisfait soit $E_{z,k}$ soit $F_{z,k}$ pour tout couple classe, file $(z,k) \in \mathcal{C} \times \mathcal{Q}$, ce qui signifie que le diagramme ne possède qu'un seul chemin.

□

# Conclusion

Nous avons présenté un modèle de réseau fermé de files d'attente multiclasses. Comme pour les réseaux fermés monoclasses, le nombre d'états à considérer est trop grand pour envisager d'appliquer directement l'algorithme de Propp et Wilson. Fort heureusement, la structure de l'espace des états permet une représentation par un diagramme somme vectorielle cumulée permettant ainsi de réduire la complexité de la représentation initialement en $O(M^{ZK})$ à $O(KM^Z)$. Suite à la définition d'un algorithme de transition agissant sur de tels diagrammes, on a proposé l'algorithme de simulation parfaite `PSDM` qui réalise la simulation parfaite à l'aide de diagrammes. Cependant, la complexité reste exponentielle en le nombre de classes $Z$ et rend donc `PSDM` inutilisable en pratique pour $Z$ grand.

Pour simplifier le modèle, on a fait l'hypothèse que chaque file possède une capacité infinie et un unique serveur. Les idées du chapitre 3 pourraient facilement être réadaptées afin de considérer plus de 1 serveur. Néanmoins, en ce qui concerne les files de capacité finies, la difficulté majeure est de s'assurer du couplage des algorithmes de simulation parfaite en montrant qu'il existe une suite couplante de transition.

Dans le chapitre 8, on présentera le *package* `M-Clones` qui implémente l'algorithme `PSDM`. Dans le chapitre suivant on s'intéresse à la simulation de réseaux possédant des synchronisations. Un tel réseau peut également être représenté par un diagramme somme vectorielle cumulée, cependant nous verrons qu'il est beaucoup plus efficace de considérer plusieurs morceaux de diagrammes pour les agréger ensuite à l'aide de ce qu'on appellera des *tables de connexions*. Ce type de diagramme sera nommé *diagramme aggloméré*. La piste des diagrammes agglomérés pourra être étudiée dans des travaux futurs afin d'améliorer la complexité de l'algorithme `PSDM`.

# 5 Extension aux réseaux avec synchronisation(s)

Dans les chapitres 3 et 4 on a présenté une nouvelle technique de simulation parfaite pour des réseaux fermés de files d'attente monoclasses et multiclasses. Cette dernière est basée sur une représentation par diagramme des ensembles d'états. Pour pouvoir appliquer cette technique, on a vu qu'il fallait disposer d'une représentation par diagramme, d'une fonction de transition et montrer qu'il existe une suite de transition qui transforment le diagramme complet en un diagramme ne possédant plus qu'un seul chemin. Dans ce chapitre on étend encore cette technique à deux modèles de réseaux comportant des synchronisations. On exhibera pour chaque modèle : un nouveau type de diagramme, une fonction de transition et une suite couplante de transitions. Les diagrammes de ce chapitre seront nommés *diagrammes agglomérés*. Ils seront composés de plusieurs morceaux de diagrammes représentant chacun un sous-réseau du modèle étudié qu'on agrégera à l'aide d'une ou plusieurs *tables de connexions*.

## 5.1 Modèle avec une synchronisation

### 5.1.1 Présentation

On considère un modèle qui contient 3 sous-réseaux de files d'attente ayant respectivement $K_1$, $K_2$ et $K_3$ files. Les sous-réseaux ne partagent aucune file d'attente. Pour tout $r \in \{1, 2, 3\}$, on note $R_r$ le $r$-ième réseau et $\mathcal{Q}(r)$ l'ensemble de ses files. Chaque sous-réseau est connexe. La figure 5.1 illustre le modèle.

L'ensemble des files du modèle est donné par $\mathcal{Q} := \mathcal{Q}(1) \cup \mathcal{Q}(2) \cup \mathcal{Q}(3)$ ainsi $|\mathcal{Q}| = K_1 + K_2 + K_3 = K$. Dans ce modèle on considère que chaque file possède un seul serveur et une capacité infinie. Les clients peuvent changer de réseaux à l'aide de deux transitions notées $T_1$ et $T_2$. Quand un couple de clients provenant des réseaux $R_1$ et $R_2$ est servi par la transition $T_1$, il est dirigé dans le réseau $R_3$. À son arrivée dans le réseau $R_3$, le couple fusionne pour ne devenir qu'un seul client dans le réseau $R_3$. Un client qui sort du réseau $R_3$ par la transition $T_2$ se dédouble et intègre les réseaux $R_1$ et $R_2$. Le nombre de clients dans chaque réseau évolue selon le nombre de fois que sont appliquées les transitions $T_1$ et $T_2$.

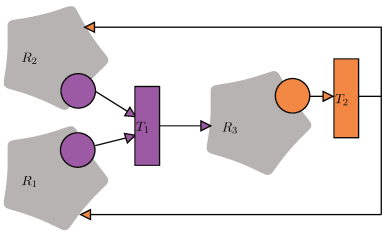



**FIGURE 5.1:** Modèle avec une synchronisation.

On note ${}^\bullet T$ l'ensemble des files servies par une transition de type $T$ et $T^\bullet$ l'ensemble des files en sortie. On a ainsi $|{}^\bullet T_1| = |T_2^\bullet| = 2$ et $|T_1^\bullet| = |{}^\bullet T_2| = 1$.

On peut distinguer 3 types de transitions :

– les transitions intra-réseau, c'est-à-dire de la file $i$ vers la file $j$ quand il existe $r \in \{1,2,3\}$ tel que $i,j \in \mathcal{Q}(r)$. On dit que ses transitions sont de type $T_0$,
– la transition $T_1$ qui s'effectue des files appartenant à ${}^\bullet T_1$ vers la file contenue dans $T_1^\bullet$,
– la transition $T_2$ qui s'effectue de la file appartenant à ${}^\bullet T_2$ vers les files contenues dans $T_2^\bullet$.

On dira que les transitions $T_1$ et $T_2$ sont respectivement de type $T_1$ et $T_2$. Les transitions de type $T_0$ s'effectuent avec un routage probabiliste. Un client de la file d'attente $i \in \mathcal{Q} \setminus \{{}^\bullet T_1 \cup {}^\bullet T_2\}$ venant d'être servi est dirigé en file $j \in \mathcal{Q}$ avec probabilité $p_{i,j}$. Les transitions $T_1$ et $T_2$ sont déterministes, ainsi pour tout $i \in {}^\bullet T$ et $j \in T^\bullet$ on a $p_{i,j} = 1$. Pour $r \in \{1,2\}$ on suppose que le réseau $R_r$ restreint aux files $\mathcal{Q}(r) \setminus {}^\bullet T_1$ est fortement connexe et qu'il existe au moins une file $i \in \mathcal{Q}(r) \setminus {}^\bullet T_1$ telle que $p_{i,j} > 0$ pour $j$ la seule file de l'ensemble $\mathcal{Q}(r) \cap {}^\bullet T_1$. De plus on suppose que le réseau $R_3$ restreint aux files $\mathcal{Q}(3) \setminus {}^\bullet T_2$ est fortement connexe et qu'il existe une file $i \in \mathcal{Q}(3) \setminus {}^\bullet T_2$ telle que $p_{i,j} > 0$ pour $j$ la seule file de l'ensemble ${}^\bullet T_2$. On note $\mathbf{P} = (p_{i,j})_{i,j \in \mathcal{Q}}$ la matrice de routage du modèle.

On note $m_r$ le nombre de clients du réseau $r$ pour une configuration donnée. Ce nombre évolue selon le nombre de fois que sont appliquées les transitions $T_1$ et $T_2$. Le nombre de clients dans l'ensemble du réseau n'est donc pas constant. Cependant, à chaque instant les nombres $m_1 + m_3$ et $m_2 + m_3$ sont constants.

## 5.1.2 Modélisation

### Espace des états

On pose $M_1 := m_1 + m_3$ et $M_2 := m_2 + m_3$. On numérote les files de chaque réseau $R_r$ de $\sum_{q=1}^{r-1} K_q + 1$ à $\sum_{q=1}^{r} K_q$. Un état du modèle peut donc être représenté par un vecteur $\mathbf{x}$ de taille $K$ tel que $x_k$ est le nombre de clients en file $k$. Un état $\mathbf{x}$ doit alors satisfaire les deux contraintes suivantes :

1. $\displaystyle\sum_{k \in \mathcal{Q}(1) \cup \mathcal{Q}(3)} x_k = M_1$,
2. $\displaystyle\sum_{k \in \mathcal{Q}(2) \cup \mathcal{Q}(3)} x_k = M_2$.

On note $\mathcal{S}$ l'espace des états du modèle. Ainsi on a

$$\mathcal{S} := \left\{ \mathbf{x} \in \mathbb{N}^K \mid \sum_{k \in \mathcal{Q}(1) \cup \mathcal{Q}(3)} x_k = M_1 \text{ et } \sum_{k \in \mathcal{Q}(2) \cup \mathcal{Q}(3)} x_k = M_2 \right\}.$$

On cherche maintenant à calculer $|\mathcal{S}|$. D'après les contraintes, le nombre de clients $m_3$ dans $R_3$ est tel que $0 \leq m_3 \leq \min(M_1, M_2)$. Pour $m_3$ fixé, on a $m_1 = M_1 - m_3$ et $m_2 = M_2 - m_3$. Or on sait qu'un réseau à $K_r$ files et $m_r$ clients dispose d'un espace d'états de cardinalité $\binom{m_r + K_r - 1}{m_r}$, ainsi on en déduit :

$$|\mathcal{S}| = \sum_{m_3=0}^{\min(M_1,M_2)} \binom{M_1 - m_3 + K_1 - 1}{M_1 - m_3} \binom{M_2 - m_3 + K_2 - 1}{M_2 - m_3} \binom{m_3 + K_3 - 1}{m_3}.$$

**Exemple 20.** *Considérerons le réseau illustré en figure 5.2 ( $K_1 = K_2 = 3$, $K_3 = 2$, i.e. $K = 8$) avec comme nombre de clients $M_1 = 2$, $M_2 = 3$. Chaque file d'attente est numérotée indépendamment de son ou ses serveurs. Un état $\mathbf{x} \in \mathcal{S}$ est donné par $\mathbf{x} = (1,0,0,0,1,1,1,0)$. Le nombre total d'états est égal à*

$$|\mathcal{S}| = 6 \times 10 \times 1 + 3 \times 6 \times 2 + 1 \times 3 \times 3 = 60 + 36 + 9 = 105.$$

$$\mathbf{P} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.3 & 0 & 0.7 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.3 & 0 & 0.7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

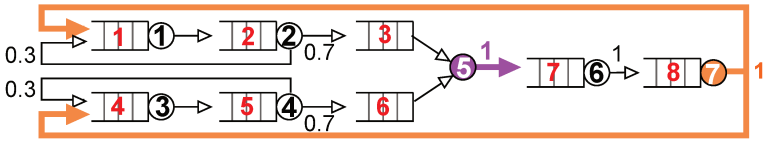



**FIGURE 5.2:** Exemple de réseaux avec une synchronisation.

### Transitions

On a vu que les transitions du modèle s'effectuent d'un ensemble de files vers un autre. Soient $\mathcal{I} \subseteq \mathcal{Q}$ et $\mathcal{J} \subseteq \mathcal{Q}$ on pose

$$T_{\mathcal{I},\mathcal{J}} := \mathbf{x} + \left(\prod_{i\in\mathcal{I}} \mathbb{1}_{\{x_i>0\}}\right)(\sum_{j\in\mathcal{J}} \mathbf{e_j} - \sum_{i\in\mathcal{I}} \mathbf{e_i}).$$

Par exemple pour une transition $T_{\mathcal{I},\mathcal{J}}$ de type $T_1$, $\mathcal{I} = {}^\bullet T_1$ est un ensemble contenant deux éléments tandis que $\mathcal{J} = T_1^\bullet$ est un singleton.

À chaque serveur est associé un seul type de transition ($T_0$, $T_1$ ou $T_2$). On numérote indépendamment les serveurs des files. On note $\mathcal{R}$ l'ensemble des serveurs du modèle. Comme pour la transition $T_1$ deux files se partagent un même serveur, on a $|\mathcal{R}| = K - 1$. On note $s_1$ le serveur qui effectue la transition de type $T_1$ et $s_2$ celui effectuant celle de type $T_2$.

### Chaîne de Markov

Pour tout $s \in \mathcal{R}$ on note $p_s = \frac{\mu_s}{\sum_{s\in\mathcal{R}} \mu_s}$ la probabilité de choisir le serveur $s$. On note $(U_n)_{n\in\mathbb{N}}$ une suite i.i.d de variables aléatoires de distribution à valeur dans $\mathcal{P}(\mathcal{Q}) \times \mathcal{P}(\mathcal{Q})$ définie telle que,

$$U_n = \begin{cases} ({}^\bullet T_1, (T_1^\bullet)) & \text{avec probabilité } p_{s_1} \ (s_1 \text{ de type } T_1) \\ ({}^\bullet T_2, (T_2^\bullet)) & \text{avec probabilité } p_{s_2} \ (s_2 \text{ de type } T_2) \\ (\{i\}, \{j\}) & \text{avec probabilité } p_s \frac{p_{i,j}}{\sum_{k\in\mathcal{Q}} p_{i,k}} \ (s \text{ de type } T_0 \text{ et } s \text{ le serveur de la file } i.) \end{cases}$$

La dynamique du système est modélisée par la chaîne de Markov $(X_n)_{n\in\mathbb{N}}$ à valeur dans $\mathcal{S}$ telle que $X_0 \in \mathcal{S}$ et

$$X_{n+1} = t_{U_n}(X_n).$$

Comme chaque sous-réseau restreint aux files étant servies par une transition de type $T_0$ est fortement connexe, il existe au moins une file de ce sous-réseau connexe qui envoie les clients vers une file disposant d'un serveur de type $T_1$, ainsi on en déduit que la chaîne

est ergodique. On cherche maintenant à appliquer l'algorithme de simulation parfaite pour échantillonner $(X_n)_{n\in\mathbb{N}}$. À cause de la cardinalité de $\mathcal{S}$ comme c'est le cas dans les chapitres 3 et 4, ceci ne peut pas être fait directement. On cherche donc, comme dans les chapitres précédents, une représentation par diagramme.

### 5.1.3 Un point de vue multiclasse

On commence par représenter l'espace des états par un diagramme conforme aux définitions du chapitre 2. On verra dans la sous-section suivante que ce point de vue n'est pas le plus efficace. Commençons par redéfinir l'espace des états par une représentation équivalente.

Dans ce modèle, on peut considérer deux types de clients : ceux pouvant circuler dans les réseaux $R_1$ et $R_3$ et ceux des réseaux $R_2$ et $R_3$. Le modèle peut alors être vu comme un réseau fermé multiclasse à $K = K_1 + K_2 + K_3$ files avec $M_1$ clients de classe $1$ et $M_2$ clients de classe $2$. Les clients de classe $1$ peuvent être présents uniquement dans les files des réseaux $R_1$ et $R_3$ et ceux de classe $2$ dans les files des réseaux $R_2$ et $R_3$. Les contraintes $m_1 + m_3 = M_1$ et $m_2 + m_3 = M_2$ exigent que dans le réseau $3$ il y ait dans chaque file autant de clients de classe $1$ que de clients de classe $2$. Par exemple

$$\mathbf{y} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

est un état valide pour le modèle. Il encode les mêmes informations que le vecteur $\mathbf{x}$ de l'exemple 20 mais en prenant deux fois plus de place mémoire. Notons $\mathbf{M} = (M_1, M_2)$. Reprenons le formalisme du chapitre 2 et redéfinissons l'espace des états. Soient $\mathcal{E} = \mathcal{G} = \mathbb{N}^2$, $\mathcal{E}_k$ pour $k \in \{1, \ldots, K\}$ tel que

$$\mathcal{E}_k = \begin{cases} \{(x, 0), x \in \mathbb{N}\} & \text{si } 1 \leq k \leq K_1 \\ \{(0, x), x \in \mathbb{N}\} & \text{si } K_1 + 1 \leq k \leq K_1 + K_2 \\ \{(x, x), x \in \mathbb{N}\} & \text{si } K_1 + K_2 + 1 \leq k \leq K, \} \end{cases}$$

et la suite $\mathcal{F} = (f^{(k)})_{k\in\mathcal{N}}$ telle que

$$f^{(k)}(\mathbf{x}) = \begin{cases} (0, 0) & \text{si } k = 0 \\ (\sum_{i=1}^{k} x_{1,i}, \sum_{i=1}^{k} x_{2,i}) & \text{sinon.} \end{cases}$$

Comme l'addition est une loi de composition associative, la suite $\mathcal{F}$ est sans mémoire. L'espace des états correspond à l'espace sans mémoire $\Omega(K, \mathbf{M})$, car on a

$$\Omega(K, \mathbf{M}) = \left\{ \mathbf{x} \in \mathcal{E}_1 \times \ldots \times \mathcal{E}_K \mid \sum_{k=1}^{K} x_{k,1} = M_1 \text{ et } \sum_{k=1}^{K} x_{k,2} = M_2 \right\} = \mathcal{S}.$$

Comme pour les réseaux multiclasses, on peut ainsi représenter tout les sous-ensembles d'états $S \subseteq \mathcal{S}$ à l'aide d'un diagramme. Par exemple, le diagramme complet de l'exemple 20 illustré par la figure 5.3 contient $64$ arcs. Dans ce diagramme, les arcs des colonnes $A(1)$, $A(2)$ et $A(3)$ sont sur le même plan. Les arcs des colonnes $A(4)$, $A(5)$ et $A(6)$ peuvent êtres divisés en $3$ plans distincts. Tandis que les arcs en colonnes $A(7)$ et $A(8)$ sont eux aussi sur un même plan. Cette représentation est redondante, on montre maintenant que l'on peut faire bien mieux. L'idée est de construire un diagramme contenant $3$ sous-diagrammes correspondant chacun à un réseau et de lier ces derniers par ce qu'on appellera une table de connexion.

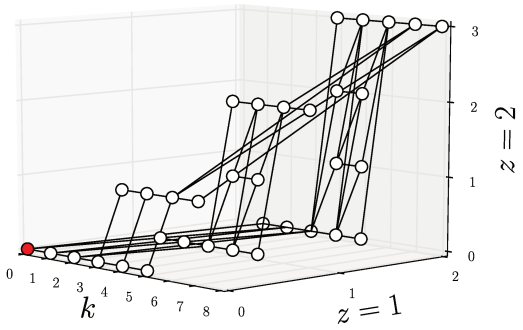



**FIGURE 5.3:** Diagramme complet $\mathcal{D}(8,(2,3))$.

### 5.1.4 Diagramme aggloméré

On appelle **sous-diagramme** un graphe représentant les états d'un sous-réseau du modèle et **table de connexion** un graphe reliant plusieurs sous-diagrammes. On appelle **diagramme aggloméré** un graphe composé de plusieurs sous-diagrammes et d'au moins une table de connexion.

Dans un sous-diagramme, comme pour un diagramme, un chemin débute par un nœud source et se termine par un nœud destination. Dans un diagramme aggloméré, on appelle **chemin** un ensemble de chemins issus de chaque sous-diagrammes qui se rejoignent dans la ou les tables de connexions. Chaque chemin d'un diagramme aggloméré correspond ainsi avec un état du modèle.

#### Description

Pour $r \in \{1,2,3\}$ on note $D_r = (\mathcal{N}_r, A_r)$ le sous-diagramme représentant le réseau $r$. Il est composé de $K_r$ colonnes numérotées de $\sum_{q=1}^{r-1} K_q + 1$ à $\sum_{q=1}^{r} K_q$ (la même numérotation que celle des files). Le triplet $(m_1, m_2, m_3)$ peut prendre $\min(M_1, M_2)$ valeurs différentes ($(M_1 - m_3, M_2 - m_3, m_3)$ pour $m_3 \in \{0, 1, \ldots, \min(M_1, M_2)\}$). Pour $r \in \{1,2\}$ le sous-diagramme possède donc 1 nœud source et $\min(M_1, M_2)$ nœuds destinations. Le sous-diagramme $D_3$ possède $\min(M_1, M_2)$ nœuds sources et 1 nœud destination. La table de connexion est composée de $\min(M_1, M_2)$ nœuds et de 2 colonnes d'arcs, on la note $D_c = (\mathcal{N}_c, A_c)$. C'est elle qui fait le lien entre les 3 sous-diagrammes. Ses colonnes portent les numéros $K+1$ et $K+2$.

On construit $D$ le diagramme aggloméré de ce modèle. Il s'agit d'un graphe composé de trois sous-diagrammes et d'une table de connexion. Il contient $K+2$ colonnes, deux nœuds sources (ceux de $D_1$ et de $D_2$) et 1 nœud destination (celui de $D_3$). Ainsi on a

$$D = (\mathcal{N}_1 \cup \mathcal{N}_2 \cup \mathcal{N}_c \cup \mathcal{N}_3, A_1 \cup A_2 \cup A_c \cup A_3).$$

Dans un diagramme aggloméré, hormis pour la table de connexion, tous les numéros de colonnes correspondent à un numéro de file.

**Exemple 21.** *La figure 5.4 représente le diagramme aggloméré complet correspondant à l'exemple 20. Si on regarde le nombre de clients réseau par réseau, il ne peut y avoir que les trois configurations suivantes* $(m_1, m_2, m_3) \in \{(0,1,2),(1,2,1),(2,3,0)\}$. *Ces configurations sont représentées par les nœuds en diamant de la table de connexion. Le diagramme aggloméré*

*complet contient* 49 *arcs et représente* 105 *états. Le chemin rouge pointillé correspond à l'état* $\mathbf{x} = (1, 0, 0, 0, 1, 1, 1, 0)$.

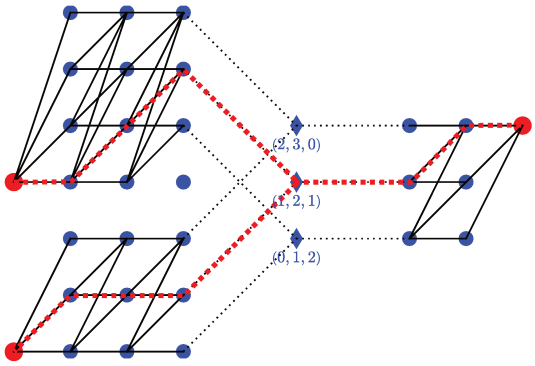



**FIGURE 5.4:** Diagramme aggloméré complet.

On peut adapter les définitions et les algorithmes sur les diagrammes, par exemple l'algorithme `CardStates` qui calcule le nombre d'états. Pour ce calcul dans un diagramme aggloméré, on commence par attribuer la valeur 1 aux deux nœuds source. Puis on calcule la valeur des nœuds de gauche à droite en faisant une addition pour les nœuds des sous-diagrammes et une multiplication pour les nœuds de la table de connexion. Le nombre d'états du réseau se lit dans le nœud destination. La figure 5.5 illustre le calcul de $|\psi(\mathcal{D})|$ pour $\mathcal{D}$ le diagramme aggloméré complet de l'exemple 20. On retrouve bien $|\psi(\mathcal{D})| = |\mathcal{S}| = 105$.

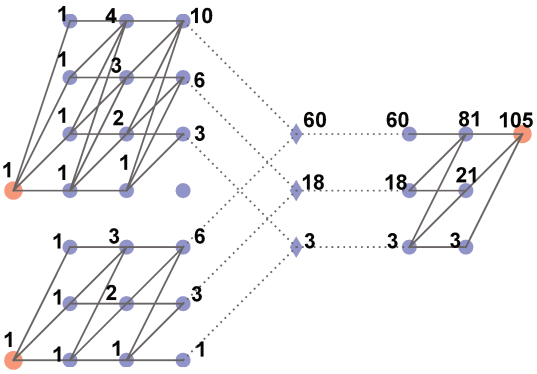



**FIGURE 5.5:** Calcul du nombre d'états dans un diagramme aggloméré.

En redéfinissant $g$ la fonction du chapitre 2 qui transforme un état en un ensemble d'arcs du diagramme aggloméré, on peut reprendre certaines définitions du chapitre 2. Les définitions des fonctions $\psi$ et $\phi$ (qui transforment respectivement un diagramme en un ensemble d'états et réciproquement) ainsi que celles des chemins dans un diagrammes $path(b)$ et $\mathcal{P}ath(B)$ restent inchangées. Nous les utiliserons par la suite.

### Algorithme de transition

Pour $\mathcal{I}, \mathcal{J} \subset \mathcal{Q}$, on définit la transition $T_{\mathcal{I},\mathcal{J}}$ sur les diagrammes agglomérés par

$$T_{\mathcal{I},\mathcal{J}} = \phi \circ t_{\mathcal{I},\mathcal{J}} \circ \psi.$$

La transition admet les propriétés suivantes.

**Lemme 22.** *Pour un ensemble d'états $S \subseteq \mathcal{S}$, un diagramme $D$, $\mathcal{I}, \mathcal{J} \subset \mathcal{Q}$ :*

*i) Si $S \subseteq \psi(D)$ alors $t_{\mathcal{I},\mathcal{J}}(S) \subseteq \psi(T_{\mathcal{I},\mathcal{J}}(D))$.*

*ii) Si $|\psi(D)| = 1$ alors $|\psi(T_{\mathcal{I},\mathcal{J}}(D))| = 1$.*

Comme pour les chapitres 3 et 4, pour réaliser la transition $T_{\mathcal{I},\mathcal{J}}$ sur un diagramme aggloméré on n'utilise pas les fonctions $\psi$ et $\phi$ mais un algorithme de transition. On présente maintenant l'algorithme 16 qui prend pour entrées $D$ un diagramme aggloméré, un couple de $(\mathcal{I}, \mathcal{J})$ et retourne $D' = T_{\mathcal{I},\mathcal{J}}(D)$.

L'algorithme 16 est divisé en trois parties selon le type de transition qu'on lui demande de réaliser ($T_0$, $T_1$ ou $T_2$). Pour une transition intra réseau ($T_0$) on a $\{\mathcal{I}\} = \{i\}$ et $\{\mathcal{J}\} = \{j\}$. La transition $T_{\mathcal{I},\mathcal{J}}$ s'effectue de la même manière que celles des réseaux monoclasses, elle correspond à la transition $T_{i,j,1}$ définie au chapitre 3 (réseau monoclasse, capacités de files infinies, un seul serveur). Les transitions extra réseau $T_1$ et $T_2$ nécessitent de traverser les nœuds diamants mais le principe reste similaire à celui des chapitres 3 et 4. On commence par identifier pour chaque $i \in \mathcal{I}$ les chemins correspondants respectivement aux états tels que $x_i = 0$ (que l'on nomme $\mathcal{E}mpty$) et $x_i > 0$ (que l'on nomme $\mathcal{T}ransit$). Puisque les files sont de capacités infinies on ne se soucie pas de $x_j$ ($\mathcal{F}ull = \emptyset$). Une fois les chemins identifiés on opère la modification des arcs contenus dans $\mathcal{T}ransit$.

Soit $A(k)_r$ la $k$ième colonne du sous-diagramme $D_r$, on définit :

- $A_{k,\mathcal{E}mpty} = \{a \in A(k)_r \mid v(a) = 0\}$,
- $A_{k,\mathcal{T}ransit} = \{a \in A(k)_r \mid v(a) > 0\}$.

On note $c_1$ la première colonne de la table de connexion et $c_2$ la deuxième colonne. La transition de type $T_1$ ne peut pas s'effectuer si aucun chemin $\mathcal{T}ransit$ des sous-diagrammes $D_1$ et $D_2$ ne se croisent dans la table de connexion, c'est-à-dire si la condition de la ligne 10 est réalisée.

### Couplage

**Lemme 23.** *Il existe une suite finie de transitions qui transforment le diagramme aggloméré complet en un diagramme ne possédant plus qu'un seul chemin.*

*Démonstration.* Considérons le réseau $R_r$ pour $r \in \{1, 2\}$. Le réseau composé des files $\mathcal{Q}(r) \setminus {}^\bullet T_1$ est fortement connexe et il existe une file dans $\mathcal{Q}(r)$ dont un des routages conduit vers la file ${}^\bullet T_1 \cap \mathcal{Q}(r)$.

**Étape 1.** On applique donc autant de fois que nécessaire des transitions de type $T_0$ dans le réseau $R_r$ afin de concentrer tous les clients de $R_r$ dans la file ${}^\bullet T_1 \cap \mathcal{Q}(r)$.

**Étape 2.** On effectue ensuite $\min(M_1, M_2)$ la transition de type $T_1$. Ainsi le réseau $R_r$ contient $M_r - \min(M_1, M_2)$ clients tous concentrés (s'il en reste) en file ${}^\bullet T_1 \cap \mathcal{Q}(r)$. L'ensemble des files $\mathcal{Q}(3) \setminus {}^\bullet T_2$ est fortement connexe et il existe une file dans $\mathcal{Q}(3)$ dont un des routages conduit vers la file ${}^\bullet T_2 \cap \mathcal{Q}(3)$.

**Étape 3.** On applique donc autant de fois que nécessaire des transitions de type $T_0$ dans le réseau $R_3$ afin de concentrer tous les clients de $R_3$ dans la file ${}^\bullet T_2$. En suivant cette succession de transitions le diagramme ne contient plus qu'un seul chemin. □

La figure 5.6 illustre trois diagrammes agglomérés. Ils correspondent aux trois étapes de la démonstration. Le plus à gauche est obtenu à partir du diagramme aggloméré complet

**Algorithme 16:** $T_{\mathcal{I},\mathcal{J}}$

**Données** : $D = (\mathcal{N}, A)$ un diagramme aggloméré
**Résultat** : $T_{\mathcal{I},\mathcal{J}}(D)$ un diagramme aggloméré

**début**
  **si** *la transition est de type* $T_0$ *(*$\mathcal{I} = \{i\} \subseteq \mathcal{Q}(r)$*,* $\mathcal{J} = \{j\} \subseteq \mathcal{Q}(r)$*)* **alors**
    $D_r \leftarrow (\mathcal{N}_r, A_r)$;
    $D'_r = (\mathcal{N}, A_r{}') \leftarrow T_{i,j,1}(D_r)$;
    $A' \leftarrow (A \setminus A_r) \cup A_r{}'$;
    **renvoyer** $D' = (\mathcal{N}, A')$;
  **si** *la transition est de type* $T_1$ *(*$\mathcal{I} = {}^\bullet T_1 = \{i_1, i_2\}$ *et* $\mathcal{J} = T_1^\bullet = \{j\}$*)* **alors**
    $\mathcal{T}ransit_1 \leftarrow \mathcal{P}ath(A_{i_1,\mathcal{T}ransit})$;
    $\mathcal{T}ransit_2 \leftarrow \mathcal{P}ath(A_{i_2,\mathcal{T}ransit})$;
    **si** $\mathcal{T}ransit_1 \cap \mathcal{T}ransit_2 \cap A(c_2) == \emptyset$ **alors**
      **renvoyer** $D$ ;
    $\mathcal{E}mpty \leftarrow \mathcal{P}ath(A_{i_1,\mathcal{E}mpty}) \cup \mathcal{P}ath(A_{i_2,\mathcal{E}mpty})$;
    $\mathcal{T}ransit \leftarrow \mathcal{P}ath(\mathcal{T}ransit_1 \cap \mathcal{T}ransit_2 \cap A(c_2))$;
    $\mathcal{T}ransit' \leftarrow \emptyset$;
    **pour** ***chaque*** $a = \big((k-1,\ell),(k,\ell')\big) \in \mathcal{T}ransit$ **faire**
      **si** $a \in A_r$ ***avec*** $r \in \{1,2\}$ **alors**
        **si** $k == i_r$ **alors**
          $\mathcal{T}ransit' \leftarrow \mathcal{T}ransit' \cup \{\big((i_r-1,\ell),(i_r,\ell'-1)\big)\}$;
        **si** $k > i_r$ ***et*** $k \in \mathcal{Q}(r)$ **alors**
          $\mathcal{T}ransit' \leftarrow \mathcal{T}ransit' \cup \{\big((k-1,\ell-1),(k,\ell'-1)\big)\}$;
      **si** $a \in A_c$ **alors**
        $\mathcal{T}ransit' \leftarrow \mathcal{T}ransit' \cup \{\big((k-1,\ell-1),(k,\ell'-1)\big)\}$;
      **si** $a \in A_3$ **alors**
        **si** $k == j$ **alors**
          $\mathcal{T}ransit' \leftarrow \mathcal{T}ransit' \cup \{\big((j-1,\ell-1),(j,\ell')\big)\}$;
        **si** $k < j$ **alors**
          $\mathcal{T}ransit' \leftarrow \mathcal{T}ransit' \cup \{\big((k-1,\ell-1),(k,\ell'-1)\big)\}$;
  **si** *la transition est de type* $T_2$ *(*$\mathcal{I} = {}^\bullet T_2 = \{i\}$ *et* $\mathcal{J} = T_2^\bullet = \{j_1, j_2\}$*)* **alors**
    $\mathcal{T}ransit \leftarrow \mathcal{P}ath(A_{i,\mathcal{T}ransit})$;
    $\mathcal{E}mpty \leftarrow \mathcal{P}ath(A_{i,\mathcal{E}mpty})$;
    $\mathcal{T}ransit' \leftarrow \emptyset$ ;
    **pour** ***chaque*** $a = \big((k-1,\ell),(k,\ell')\big) \in \mathcal{T}ransit$ **faire**
      **si** $a \in A_3$ **alors**
        **si** $k == i$ **alors**
          $\mathcal{T}ransit' \leftarrow \mathcal{T}ransit' \cup \{\big((i-1,\ell+1),(i,\ell')\big)\}$;
        **si** $k < i$ **alors**
          $\mathcal{T}ransit' \leftarrow \mathcal{T}ransit' \cup \{\big((k-1,\ell+1),(k,\ell'+1)\big)\}$;
      **si** $a \in A_c$ **alors**
        $\mathcal{T}ransit' \leftarrow \mathcal{T}ransit' \cup \{\big((k-1,\ell+1),(k,\ell'+1)\big)\}$;
      **si** $a \in A_r$ ***avec*** $r \in \{1,2\}$ **alors**
        **si** $k == j_r$ **alors**
          $\mathcal{T}ransit' \leftarrow \mathcal{T}ransit' \cup \{\big((j_r-1,\ell),(j_r,\ell'+1)\big)\}$;
        **si** ***et*** $k \in \mathcal{Q}(r)$ **alors**
          $\mathcal{T}ransit' \leftarrow \mathcal{T}ransit' \cup \{\big((k-1,\ell+1),(k,\ell'+1)\big)\}$;
  $A' \leftarrow \mathcal{T}ransit' \cup \mathcal{E}mpty$;
  **renvoyer** $D' = (\mathcal{N}, A')$.

en effectuant les transitions $T^3_{5,6} \circ T^3_{4,5} \circ T^2_{2,3} \circ T^2_{1,2}$. Les clients des réseaux $R_1$ et $R_2$ sont concentrés respectivement dans les files 3 et 6. Le diagramme du milieu est obtenu à partir de celui de gauche en effectuant 2 fois la transition de type $T_1$, c'est-à-dire $T^2_{3,6,7}$. Pour finir,

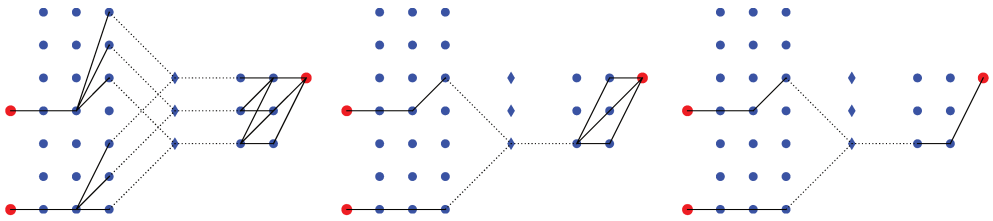


**FIGURE 5.6:** Couplage du diagramme aggloméré

celui de droite est obtenu en effectuant $T^2_{7,8}$, il ne contient qu'un seul chemin. Ce chemin correspond à l'état $(0,0,0,0,0,1,0,2) \in \mathcal{S}$.

## 5.2 Modèle avec plusieurs synchronisations

### 5.2.1 Présentation et modélisation

On considère un modèle qui contient $4$ sous-réseaux de files d'attente qui ne partagent pas de file d'attente. Les notations sont identiques à celles du modèle à une synchronisation. Chaque file possède un seul serveur et une capacité infinie. Les clients ne peuvent pas changer de réseau. Il y a deux types de transitions. Les transitions de type $T_0$ s'effectuent à l'intérieur d'un même réseau avec des routages probabilistes. Les transitions $T_{31}$, $T_{32}$ et $T_{33}$ dites de type $T_3$, s'effectuent quant à elle pour un couple de clients et ont un routage déterministe. Une transition de type $T_3$ sert un couple de clients provenant de deux réseaux distincts. Une fois servi, le couple se sépare et chaque client retrouve son réseau initial. La figure 5.7 illustre le modèle.

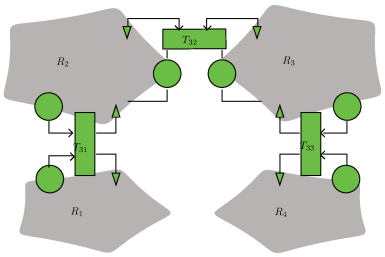



**FIGURE 5.7:** Modèle avec plusieurs synchronisations.

On note ${}^\bullet T_3 = {}^\bullet T_{31} \cup {}^\bullet T_{32} \cup {}^\bullet T_{33}$. Pour $r \in \{1,2,3,4\}$ on suppose que le réseau $R_r$ restreint aux files de $\mathcal{Q}(r) \setminus {}^\bullet T_3$ est fortement connexe et qu'il existe pour chaque file $j \in \mathcal{Q}(r) \cap {}^\bullet T_3$ au moins une file $i \in Q(r) \setminus {}^\bullet T_3$ telle que $p_{i,j} > 0$. De plus on suppose ${}^\bullet T_3 \cap T_3^\bullet = \emptyset$.

Le nombre de clients dans l'ensemble du réseau est constant, il est égal à $M = m_1 + m_2 + m_3 + m_4$. On numérote les files de chaque réseau $R_r$ de $\sum_{q=1}^{r-1} K_q + 1$ à $\sum_{q=1}^{r} K_q$. Un état du modèle peut donc être représenté par un vecteur $\mathbf{x}$ de taille $K$ tel que pour tout $r \in \{1,2,3,4\}$ on a $\sum\limits_{k \in \mathcal{Q}(r)} x_k = m_r$.

L'espace des états du modèle est donné par,

$$\mathcal{S} := \left\{ \mathbf{x} \in \mathbb{N}^K \mid \sum_{k \in \mathcal{Q}(r)} x_k = m_r \right\}.$$

On en déduit ainsi :

$$|\mathcal{S}| = \prod_{r=1}^{4} \binom{m_r + K_r - 1}{m_r}.$$

**Exemple 22.** *Prenons le réseau illustré en figure 5.8 ( $K_1 = K_4 = 3$, $K_2 = K_3 = 4$, i.e. $K = 14$) avec comme nombre de clients $m_r = 1$ pour tous les sous-réseaux ($M = 4$). Les files sont numérotées indépendamment des serveurs. Un état $\mathbf{x} \in \mathcal{S}$ est donné par $\mathbf{x} = (0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1)$. Le nombre total d'états est égal à $|\mathcal{S}| = 3 \times 4 \times 4 \times 3 = 144$.*

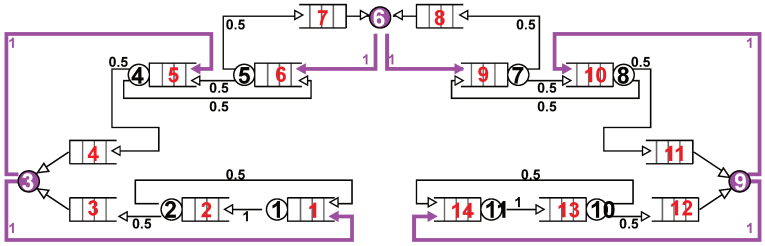


**FIGURE 5.8:** Exemple de réseaux avec plusieurs synchronisations.

On note $(U_n)_{n\in\mathbb{N}}$ une suite i.i.d de variables aléatoires de distribution à valeur dans $\mathcal{P}(\mathcal{Q}) \times \mathcal{P}(\mathcal{Q})$ définie telle que

$$U_n = \begin{cases} (\{i\}, \{j\}) & \text{avec probabilité } p_s \frac{p_{i,j}}{\sum_{k\in\mathcal{Q}} p_{i,k}} \ (s \text{ de type } T_0 \text{ et s le serveur de la file } i) \\ ({}^\bullet T_T, (T_T^\bullet)) & \text{avec probabilité } p_{s_T} \ (s_T \text{ de type } T_3). \end{cases}$$

On pose $(X_n)_{n\in\mathbb{N}}$ la chaîne de Markov ergodique à valeur dans $\mathcal{S}$ telle que $X_0 \in \mathcal{S}$ et

$$X_{n+1} = t_{U_n}(X_n).$$

## 5.2.2 Diagramme aggloméré

### Description

Le diagramme aggloméré est composé de $4$ sous-diagrammes et de trois tables de connexion. Pour $r \in \{1, 2, 3, 4\}$ les colonnes de chaque sous-diagramme $D_r = (\mathcal{N}_r, A_r)$ sont numérotées de $\sum_{q=1}^{r-1} K_q + 1$ à $\sum_{q=1}^{r} K_q$. Les sous-diagrammes contiennent chacun un nœud source et un nœud destination. Chaque table de connexion correspond à une transition de type $T_3$ et fait le lien entre deux sous-réseaux, chaque table de connexion contient un nœud et deux arcs. On note $C_{r,r'} = (\mathcal{N}_{r,r'}, A_{r,r'})$ la table de connexion faisant le lien entre les réseaux $R_r$ et $R_{r'}$, le nœud de cette table à pour valeur $(m_r, m_{r'})$. Le diagramme aggloméré correspondant à ce modèle est donné par

$$D = (\mathcal{N}_1 \cup \mathcal{N}_{1,2} \cup \mathcal{N}_2 \cup \mathcal{N}_{2,3} \cup \mathcal{N}_3 \cup \mathcal{N}_{3,4} \cup \mathcal{N}_4, A_1 \cup A_{1,2} \cup A_2 \cup A_{2,3} \cup A_3 \cup A_{3,4} \cup A_4).$$

**Exemple 23.** *La figure 5.9 représente le diagramme aggloméré complet correspondant à l'exemple 22. Le chemin en pointillé (et rouge) correspond à l'état $\mathbf{x} = (0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1)$. Les sous-réseaux sont représentés de gauche à droite, dans l'ordre croissant : $R_1$ est représenté par le sous-diagramme le plus à gauche, le réseau $R_4$ par celui le plus à droite. Comme chaque sous-réseau ne peut contenir qu'un seul client, les sous-diagrammes ne peuvent avoir que deux types d'arcs ceux de valeur $0$ (arcs horizontaux) et ceux de valeur $1$. Chaque chemin dans chaque sous-diagramme ne possède donc qu'un seul arc de valeur $1$, les autres étant de valeur $0$.*

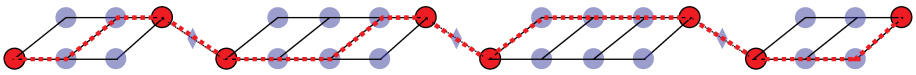


**FIGURE 5.9:** Diagramme aggloméré complet.

Remarquons que dans ce modèle tous les sous-diagrammes correspondent exactement à des diagrammes au sens du chapitre 3.

### Algorithme de transition

Pour le diagramme aggloméré décrit ci-dessus, on distingue 2 types de transitions : $T_0$ et $T_3$.

Comme pour le modèle précédent, une transition de type $T_0$ s'effectue à l'intérieur d'un même sous-réseau et correspond algorithmiquement à une transition dans un diagramme au sens du chapitre 3 (monoclasse, capacité infinie, un seul serveur).

Définissons maintenant les transitions de type $T_3$. Une transition $T$ de type $T_3$ implique deux sous-diagrammes $D_r$ et $D_{r'}$ reliés par la table de connexion $C_{r,r'}$. Posons $\mathcal{I} := {}^\bullet T_T := \{i_r, i_{r'}\}$ et $\mathcal{J} := T_T^\bullet := \{j_r, j_{r'}\}$. Comme dans le modèle précédent, on définit les deux sous-ensemble d'arcs $\mathcal{E}mpty$ et $\mathcal{T}ransit$ (non nécessairement disjoints) comme :

- $\mathcal{E}mpty = \mathcal{P}ath(A_{i_r,\mathcal{E}mpty}) \cup \mathcal{P}ath(A_{i_{r'},\mathcal{E}mpty})$ avec $A_{i_r,\mathcal{E}mpty} := \{a \in A(i_r) \mid v(a) = 0\}$,
- $\mathcal{T}ransit = \mathcal{P}ath(A_{i_r,\mathcal{T}ransit}) \cap \mathcal{P}ath(A_{i_{r'},\mathcal{T}ransit})$ avec $A_{i_r,\mathcal{T}ransit} := \{a \in A(i_r) \mid v(a) > 0\}$.

On dit que le nœud diamant (celui de la table de connexion) est dans l'état $\mathcal{E}mpty$ et respectivement $\mathcal{T}ransit$ si il possède au moins un arc entrant (ou sortant) appartenant à l'ensemble $\mathcal{E}mpty$ et respectivement $\mathcal{T}ransit$. Ainsi, le nœud de la table de connexion peut être dans trois états distincts $\mathcal{E}mpty$, $\mathcal{E}mpty \cup \mathcal{T}ransit$ ou $\mathcal{T}ransit$. Si le nœud diamant est dans l'état $\mathcal{E}mpty$ alors la transition de type $T_3$ n'affecte pas le diagramme. S'il est dans l'état $\mathcal{T}ransit$ alors les transitions $T_{i_r,j_r}$ et $T_{i_{r'},j_{r'}}$ s'appliquent respectivement sur les sous-diagrammes $D_r$ et $D_{r'}$. Enfin, si le nœud diamant est dans l'état $\mathcal{E}mpty \cup \mathcal{T}ransit$ alors on conserve les arcs du diagramme $D$ et on ajoute les arcs du diagramme qui aurait été obtenu si le nœud diamant avait été dans l'état $\mathcal{T}ransit$. L'algorithme 17 décrit la transition $T_{\mathcal{I},\mathcal{J}}$ pour le modèle à plusieurs synchronisations.

### Couplage

**Lemme 24.** *Il existe une suite finie de transitions qui transforment le diagramme aggloméré complet en un diagramme ne possédant plus qu'un seul chemin.*

*Démonstration.*
**Étape 1.** On rassemble tout les clients du réseau $R_1$ dans la file $\mathcal{Q}(1) \cap {}^\bullet T_{31}$ et tout les clients de $R_4$ dans $\mathcal{Q}(4) \cap {}^\bullet T_{33}$ en appliquant des transitions de type $T_0$. Ceci est possible car on a supposé les réseaux composés des files $\mathcal{Q}(1) \setminus {}^\bullet T_{31}$ et $\mathcal{Q}(4) \setminus {}^\bullet T_{33}$ sont fortement connexes et qu'il existe des routages de $\mathcal{Q}(1) \setminus {}^\bullet T_{31}$ vers $\mathcal{Q}(1) \cap {}^\bullet T_{31}$ et de $\mathcal{Q}(4) \setminus {}^\bullet T_{33}$ vers $\mathcal{Q}(4) \cap {}^\bullet T_{33}$.

**Étape 2.** Le but est de vider les files $T_{31}^\bullet \cap \mathcal{Q}(2)$ et $T_{33}^\bullet \cap \mathcal{Q}(3)$. Pour ce faire, on applique autant de fois que nécessaire les transitions $T_{31}$ et $T_{33}$ suivies de l'étape 1.

**Étape 3.** On rassemble les clients des sous-réseaux $R_2$ et $R_3$ dans les files ${}^\bullet T_{32}$ en appliquant des transitions de type $T_0$. Ceci est possible car on a supposé les réseaux composés des files

**Algorithme 17:** $T_{\mathcal{I},\mathcal{J}}$

**Données :** $D = (\mathcal{N}, A)$ un diagramme aggloméré
**Résultat :** $T_{\mathcal{I},\mathcal{J}}(D)$ un diagramme aggloméré

**début**
  **si** *la transition est de type* $T_0$ *(*$\mathcal{I} = \{i\} \subseteq \mathcal{Q}(r)$*,* $\mathcal{J} = \{j\} \subseteq \mathcal{Q}(r)$*)* **alors**
    $(\mathcal{N}, B) \leftarrow T_{i,j,1}(D_r)$;
    $A' \leftarrow (A \setminus A_r) \cup B$;
    **renvoyer** $D' = (\mathcal{N}, A')$;
  **si** *la transition est de type* $T_3$ *(* $\mathcal{I} = \{i_r, i_{r'}\}$ *et* $\mathcal{J} = \{j_r, j_{r'}\}$*)* **alors**
    $\mathcal{E}mpty \leftarrow \mathcal{P}ath(A_{i_r,\mathcal{E}mpty}) \cup \mathcal{P}ath(A_{i_{r'},\mathcal{E}mpty})$;
    $\mathcal{T}ransit \leftarrow \mathcal{P}ath(A_{i_r,\mathcal{T}ransit}) \cap \mathcal{P}ath(A_{i_{r'},\mathcal{T}ransit})$;
    **si** $\mathcal{T}ransit == \emptyset$ **alors**
      **renvoyer** $D$ ;
    $(\mathcal{N}, B_r) \leftarrow T_{i_r,j_r,1}(D_r)$;
    $(\mathcal{N}, B_{r'}) \leftarrow T_{i_{r'},j_{r'},1}(D_{r'})$;
    **si** $\mathcal{T}ransit \cap A_{r,r'}(1) == A_{r,r'}(1)$ **alors**
      $A' \leftarrow (A \setminus (A_r \cup A_{r'})) \cup B \cup B'$;
    **sinon**
      $A' \leftarrow A \cup B \cup B'$;
    **renvoyer** $D' = (\mathcal{N}, A')$.

$\mathcal{Q}(2) \setminus {}^\bullet T_{32}$ et $\mathcal{Q}(3) \setminus {}^\bullet T_{32}$ sont fortement connexes et qu'il existe des routages de $\mathcal{Q}(2) \setminus {}^\bullet T_{32}$ vers $\mathcal{Q}(2) \cap {}^\bullet T_{32}$ et de $\mathcal{Q}(3) \setminus {}^\bullet T_{32}$ vers $\mathcal{Q}(3) \cap {}^\bullet T_{32}$.

En appliquant cette succession d'étapes, le diagramme ne contient plus qu'un seul chemin.

□

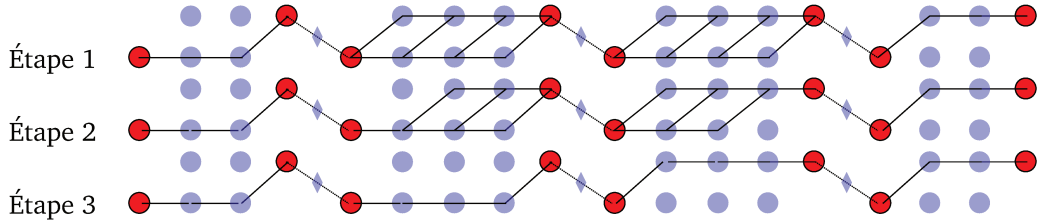



**FIGURE 5.10:** Couplage du diagramme aggloméré.

# Conclusion

Pour les deux modèles présentés dans ce chapitre, on a décrit un algorithme de transition et prouvé qu'il existe une suite de transitions qui transforment le diagramme aggloméré complet en un diagramme ne possédant qu'un seul chemin. On a fait comme hypothèse dans les deux modèles que l'on dispose de sous-réseaux fortement connexes. On a montré par ailleurs que l'on peut utiliser la technique de simulation parfaite avec diagramme sur ces deux modèles. Ces résultats laissent penser qu'il est possible d'étendre la technique à d'autres type de réseaux possédant une ou plusieurs synchronisations. Les points essentiels sont d'exhiber un diagramme aggloméré puis de monter qu'il existe une suite couplante de transitions pour le diagramme aggloméré. La généralisation de cette technique pourra faire l'objet de futures recherches.

# Deuxième partie

Méthode de Boltzmann

# Échantillonnage de Boltzmann

# 6

L'algorithme de simulation parfaite permet l'échantillonnage sans biais d'une chaîne de Markov ergodique. Dans cette nouvelle partie, on intéresse à une autre technique d'échantillonnage sans biais : la méthode de Boltzmann. Cette dernière fut introduite en 2004 par Duchon, Flajolet, Louchard et Schaeffer [24] et a pour but la génération uniforme d'éléments de même taille appartenant à une même classe combinatoire.

La première section introduit des classes combinatoires et des fonctions génératrices en reprenant très brièvement les résultats des chapitres I et III du livre [27], et présente ensuite les générateurs de Boltzmann [24]. Dans la seconde section on montre que les états d'un réseau fermé de files d'attente monoclasses peuvent être décrit par une classe combinatoire. On propose ensuite deux générateurs de Boltzmann afin d'échantillonner la distribution d'un réseau de Gordon et Newell.

## 6.1 La méthode de Boltzmann

### 6.1.1 Classes combinatoire et fonctions génératrice

#### Fonction génératrice ordinaire

Une **classe combinatoire** est un couple $(\mathcal{A}, |.|)$ où $\mathcal{A}$ désigne un ensemble et $|.|$ est une fonction

$$\begin{array}{rccc} |.| & : & \mathcal{A} & \to & \mathbb{N} \\ & & a & \mapsto & |a| \end{array}$$

appelée **fonction taille**. Cette fonction est telle qu'il y a un nombre fini d'éléments pour une taille donnée. Pour $n \in \mathbb{N}$, on note $\mathcal{A}_n$ l'ensemble des éléments de $\mathcal{A}$ de taille $n$ et $a_n$ la cardinalité de $\mathcal{A}_n$, on a ainsi

$$\mathcal{A}_n := \{\alpha \in \mathcal{A} \text{ tels que } |\alpha| = n\} \quad \text{et} \quad a_n := |\mathcal{A}_n| < \infty.$$

Les ensembles $\mathcal{A}_n$ pour $n \in \mathbb{N}$, forment une partition de $\mathcal{A}$ : $\mathcal{A} = \cup_{n\in\mathbb{N}}\mathcal{A}_n$. À chaque classe combinatoire $(\mathcal{A}, |.|)$ est associée la série formelle $A(z)$, appelée **fonction génératrice** (ordinaire) et définie par

$$A(z) = \sum_{\alpha\in\mathcal{A}} z^{|\alpha|}. \tag{6.1}$$

En regroupant les éléments $\alpha \in \mathcal{A}$ par taille, on obtient la formulation équivalente suivante pour la fonction génératrice

$$A(z) = \sum_{n\in\mathbb{N}} a_n z^n. \tag{6.2}$$

On dit que la variable $z$ **marque la taille** de la série génératrice $A(z)$ et on note $[z^n]A(z) := a_n$ qu'on appelle **opération d'extraction** du coefficient de $z^n$ de la fonction génératrice $A(z)$. Pour le moment, $z$ est en paramètre formel et on ne s'intéresse pas aux propriétés analytiques de $A(z)$.

**Exemple 24.** *L'ensemble $\mathcal{A}$ de la figure 6.1 est constitué de $6$ polygones : $2$ verts et $4$ rouges. Considérons la fonction taille qui à chaque polygone associe son nombre de sommets, alors $A_s(z) = z^3 + 2z^4 + 2z^5 + z^8$. Considérons maintenant une autre fonction taille : celle associant la valeur $1$ pour un polygone vert et $3$ sinon. Dans ce cas, $A_c(z) = 2z+4z^3$ et il y a $[z^3]A_c(z) = 4$ polygones rouges.*

**FIGURE 6.1:** Ensemble $\mathcal{A}$.

**Exemple 25** (Mots finis sur un alphabet binaire)**.** *Considérons $\mathcal{A} = \{0,1\}^\star$ l'ensemble des mots finis sur un alphabet binaire muni d'une fonction taille correspondant à la longueur du mot. Pour chaque $n \in \mathbb{N}$, $a_n = 2^n$ et donc $A(z) = \sum_{n\in\mathbb{N}} 2^n z^n = \frac{1}{1-2z}$.*

On dit que deux classes classes combinatoires $(\mathcal{A}, |.|_\mathcal{A})$ et $(\mathcal{B}, |.|_\mathcal{B})$ sont **combinatoirement isomorphes** lorsqu'il existe une bijection $f : \mathcal{A} \to \mathcal{B}$ telle que pour tout $\alpha \in \mathcal{A}$ on a $|\alpha|_\mathcal{A} = |f(\alpha)|_\mathcal{B}$. Dans ce cas, on a donc $A(z) = B(z)$ et on note $\mathcal{A} \cong \mathcal{B}$,

### Fonction génératrice ordinaire multivariée

Lorsque l'on souhaite mesurer plusieurs critères sur les éléments d'une classe combinatoire, par exemple dénombrer les triangles rouges de l'exemple 24, on utilise les fonctions multivariées. Une fonction taille de dimension $d \in \mathbb{N}$ (pour $d$ critères) est de la forme

$$\begin{array}{rccc} |.| & : & \mathcal{A} & \to & \mathbb{N}^d \\ & & \alpha & \mapsto & (|\alpha|_1, \ldots, |\alpha|_d), \end{array}$$

où les $|.|_1, |.|_2, \ldots, |.|_d$ sont des tailles au sens des fonctions génératrices monovariées, c'est-à-dire qu'elles ne mesurent qu'un seul critère. Pour un vecteur taille $\mathbf{n} = (n_1, n_2, \ldots, n_d) \in \mathbb{N}^d$, $\mathcal{A}_\mathbf{n}$ est l'ensemble des éléments de $\mathcal{A}$ de taille $n_k$ pour chaque $|.|_k$. On fait l'hypothèse que pour tout $\mathbf{n}$, $\mathcal{A}_\mathbf{n}$ est fini. On note $a_\mathbf{n}$ sa cardinalité, ainsi on a

$$\mathcal{A}_\mathbf{n} := \{\alpha \in \mathcal{A} \text{ tels que } |\alpha|_1 = n_1, \ldots, |\alpha|_d = n_d\} \quad \text{et} \quad |\mathcal{A}_\mathbf{n}| := a_\mathbf{n} < \infty.$$

La fonction génératrice $A(\mathbf{z})$ d'une classe combinatoire $(\mathcal{A}, |.|)$ est dite multivariée et est définie par

$$A(\mathbf{z}) = A(z_1, \ldots, z_d) = \sum_{\alpha\in\mathcal{A}} \prod_{k=1}^{d} z_k^{|\alpha|_k} = \sum_{\mathbf{n}=(n_1,n_2,\ldots,n_d)\in\mathbb{N}} \left( a_\mathbf{n} \prod_{k=1}^{d} z_k^{n_k} \right). \tag{6.3}$$

La variable $z_k$ **marque la $k$-ième taille** dans la fonction génératrice. On note $[z_k^n]A(\mathbf{z})$ l'opération d'extraction du coefficient de $z_k^n$ de la fonction génératrice $A(\mathbf{z})$, c'est-à-dire :

$$[z_k^n]A(\mathbf{z}) = \sum_{\mathbf{n}=(n_1,\ldots,n_{k-1},n,n_{k+1},\ldots,n_d)\in\mathbb{N}^d} \left( a_\mathbf{n} \prod_{j=1,j\neq k}^{d} z_j^{n_j} \right).$$

**Exemple 26.** *On considère une nouvelle fois l'ensemble de la figure 6.1. Si la variable $z$ marque le nombre de sommets et $u$ la couleur ($1$ pour vert et $3$ pour rouge), alors*

$$A(z,u) = z^3u^3 + z^4u^3 + z^4u + z^5u^3 + z^5u + z^8u^3.$$

*La fonction génératrice qui compte le nombre de sommets des polygones rouges est alors donnée par*

$$[u^3]A(z,u) = z^3 + z^4 + z^5 + z^8.$$

### 6.1.2 Constructions basiques

On voudrait maintenant obtenir des classes combinatoires à partir de classes plus simples. Les résultats présentés dans ce paragraphe sont ceux du chapitre II de [27] pour des fonctions génératrices monovariées. Ces derniers se généralisent dans le cas multivarié et font l'objet du Chapitre III de [27]. Pour alléger les notations, on désignera désormais $\mathcal{A}$ la classe combinatoire $(\mathcal{A}, |.|)$. En cas d'ambiguïté sur la fonction taille, elle sera notée $|.|_{\mathcal{A}}$.

#### Union disjointe

Considérons deux classes $\mathcal{B}$ et $\mathcal{C}$ telles que $\mathcal{B} \cap \mathcal{C} = \emptyset$. On note $\mathcal{A} = \mathcal{B} + \mathcal{C}$ **l'union disjointe** des classes $\mathcal{B}$ et $\mathcal{C}$. La fonction taille $|.|_{\mathcal{A}}$ attribue aux éléments de $\mathcal{A}$ la taille au sens de leur classe d'origine, c'est-à-dire

$$|\alpha|_{\mathcal{A}} = \begin{cases} |\alpha|_{\mathcal{B}} & \text{si } \alpha \in \mathcal{B} \\ |\alpha|_{\mathcal{C}} & \text{si } \alpha \in \mathcal{C}. \end{cases}$$

De cette définition, on déduit que pour $n \in \mathbb{N}$ fixé, $a_n = b_n + c_n$, et donc

$$A(z) = \sum_{\alpha \in \mathcal{A}} z^{|\alpha|_{\mathcal{A}}} = \sum_{\alpha \in \mathcal{B}} z^{|\alpha|_{\mathcal{B}}} + \sum_{\alpha \in \mathcal{C}} z^{|\alpha|_{\mathcal{C}}} = B(z) + C(z).$$

#### Produit cartésien

On note $\mathcal{A} = \mathcal{B} \times \mathcal{C}$ **le produit cartésien** des classes $\mathcal{B}$ et $\mathcal{C}$. Pour $\alpha \in \mathcal{A}$, il existe un unique $\beta \in \mathcal{B}$ et un unique $\gamma \in \mathcal{C}$ tels que $\alpha = (\beta, \gamma)$. La fonction taille $|.|_{\mathcal{A}}$ attribue à $\alpha$ la somme des tailles respectives de $\beta$ et $\gamma$ :

$$|\alpha|_{\mathcal{A}} = |\beta|_{\mathcal{B}} + |\gamma|_{\mathcal{C}}.$$

De cette définition, on a pour $n \in \mathbb{N}$ fixé, $a_n = \sum_{k=0}^{n} b_k c_{n-k}$, et donc

$$A(z) = \sum_{\alpha \in \mathcal{A}} z^{|\alpha|_{\mathcal{A}}} = \sum_{\beta \in \mathcal{B}, \gamma \in \mathcal{C}} z^{|\beta|_{\mathcal{B}} + |\gamma|_{\mathcal{C}}} = B(z)C(z).$$

#### Classes atomiques

On note $\mathcal{E}$ une classe contenant un unique élément de taille $0$, sa fonction génératrice est alors $E(z) = 1$. Cette classe est dite **neutre** (ou atomique de taille 0). Elle ne contient qu'un seul élément qui est de taille 0. Pour toute classe combinatoire $\mathcal{A}$ on a : $\mathcal{A} \cong \mathcal{E} \times \mathcal{A} \cong \mathcal{A} \times \mathcal{E}$.

On note $\mathcal{Z}$ une classe ne contenant un unique élément de taille $1$, sa fonction génératrice est alors $Z(z) = z$. Cette classe est dite **atomique de taille 1** car elle ne contient qu'un seul élément. Remarquons que les classes neutres sont isomorphes entre elles, ainsi que les

classes atomiques de taille 1. On parlera alors de *la* classe neutre $\mathcal{E}$ et de *la* classe atomique $\mathcal{Z}$.

### Séquence

Soit $\mathcal{B}$ une classe combinatoire ne contenant aucun élément de taille $0$, on appelle **séquence** issue de la classe combinatoire $\mathcal{B}$, la classe $\textsc{seq}(\mathcal{B})$ définie par

$$\mathcal{A} := \textsc{seq}(\mathcal{B}) = \mathcal{E} + \mathcal{B} + \mathcal{B} \times \mathcal{B} + \mathcal{B} \times \mathcal{B} \times \mathcal{B} + \cdots .$$

Une séquence de $\textsc{seq}(\mathcal{B})$ est composée de $k$-uplets avec $k \in \mathbb{N}$. Chaque $k$-uplet correspond à un élément de la classe $\mathcal{B}$. Ainsi, pour tout $\alpha = (\beta_1, \ldots, \beta_k) \in \textsc{seq}(\mathcal{B})$, la taille de $\alpha$ est définie par la formule

$$|\alpha|_{\mathcal{A}} = \sum_{i=1}^{k} |\beta_i|_{\mathcal{B}}.$$

En appliquant les formules précédentes, on obtient

$$A(z) = 1 + B(z) + B(z)^2 + B(z)^3 + \cdots = \sum_{n=0}^{\infty} B(z)^n = \frac{1}{1 - B(z)}. \tag{6.4}$$

Remarquons que la classe des séquences peut également être définie de manière récursive comme l'union disjointe de l'élément neutre et du produit de $\mathcal{B}$ avec elle-même par

$$\mathcal{A} = \mathcal{E} + \mathcal{B} \times \mathcal{A}. \tag{6.5}$$

À partir de la spécification (6.5) on obtient $A(z) = 1 + B(z)A(z)$, ce qui est en accord avec la formule (6.4).

**Exemple 27** (Mots finis sur un alphabet binaire)**.** *Reprenons la classe $\mathcal{A}$ de l'exemple 25, celle des mots finis sur un alphabet binaire. Elle peut aussi être définie à partir des classes atomiques $\mathcal{B} := \{0\}$ et $\mathcal{C} := \{1\}$ par $\mathcal{A} = \textsc{seq}(\mathcal{B} + \mathcal{C})$. Ainsi $B(z) = C(z) = z$ et on retrouve $A(z) = \frac{1}{1-(B(z)+C(z))} = \frac{1}{1-2z}$, ce qui correspond bien à la formule de l'exemple 25.*

**Exemple 28** (Arbre binaire)**.** *On considère la classe des arbres binaires $\mathcal{B}$ dans laquelle la fonction taille compte les nœuds internes (i.e. les nœuds qui ne sont pas des feuilles). Un arbre peut se définir de manière récursive : en effet, un arbre est soit une feuille soit un nœud interne possédant deux arbres fils. On en déduit ainsi une expression pour $\mathcal{B}$ et sa fonction génératrice $B(z)$ :*

$$\mathcal{B} = \mathcal{E} + \mathcal{Z} \times \mathcal{B} \times \mathcal{B} \qquad \textit{et} \qquad B(z) = 1 + zB(z)^2.$$

*Ainsi on obtient*

$$B(z) = \frac{1 - \sqrt{1 - 4z}}{2z} \qquad \textit{et} \qquad b_n = \frac{1}{n+1}\binom{2n}{n}.$$

*Le coefficient $b_n$ correspond au $n$-ième nombre de Catalan. On obtient $b_n$ à partir de la spécification de $B(z)$ en procédant par récurrence.*
*Un arbre binaire à $n$ nœuds internes possède $n + 1$ feuilles et donc $2n + 1$ nœuds au total. La figure 6.2 illustre les $5 (= b_3)$ arbres binaires à $3$ nœuds internes (et donc $7$ nœuds au total).*

*Considérons maintenant une autre fonction taille pour les arbres binaires. Soit $\mathcal{C}$ la classe combinatoire des arbres binaire dans laquelle on compte tous les nœuds (indistinctement nœuds internes ou feuilles). On a alors :*

$$\mathcal{C} = \mathcal{Z} + \mathcal{Z} \times \mathcal{C} \times \mathcal{C} \quad \textit{et} \quad C(z) = z\left(1 + C(z)^2\right).$$

*On retrouve $c_{2n+1} = b_n$ le nombre d'arbres binaires à $2n+1$ nœuds (ou de manière équivalente à $n$ nœuds internes).*

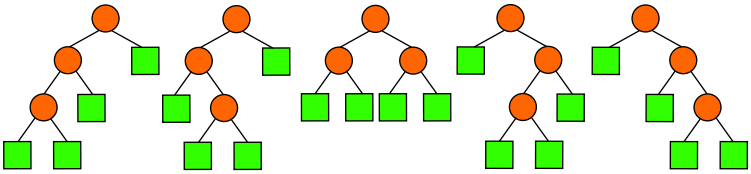


**FIGURE 6.2:** Arbres binaires à 7 nœuds (3 nœuds internes et 4 feuilles)

D'autres constructions basiques sont définies dans [24] : les cycles, les ensembles (c'est-à-dire les ensembles au sens classique du terme) et les multi-ensembles. Ces derniers feront l'objet du chapitre 7.

### 6.1.3 Générateur de Boltzmann

**Définition 12** ([24]). *On appelle **générateur de Boltzmann** de paramètre $x$ de la classe combinatoire $(\mathcal{A}, |.|)$ un algorithme probabiliste qui produit $\alpha \in \mathcal{A}$ avec probabilité $\mathbb{P}_x(\alpha) = \frac{x^{|\alpha|}}{A(x)}$. On le note $\Gamma[\mathcal{A}](x)$.*

Le paramètre $x$ doit être choisi inférieur au rayon de convergence de la fonction génératrice $A(z)$ pour que $A(x)$ soit bien définie. On note $R_A$ la singularité dominante de $A(z)$, c'est-à-dire, pour $x \in ]0, R_A[$ on a $A(x) < \infty$. Ceci est possible car les coefficients de la série génératrices sont positifs. Dans [50] Pivoteau *et al.* donnent une méthode pour calculer $A(x)$ basée sur l'itération de Newton. L'algorithme produisant $A(x)$ est appelé **oracle**.

La définition des générateurs de Boltzmann implique que deux objets de même taille ont la même chance d'être générés. La taille de l'objet produit par un générateur de Boltzmann est aléatoire. Cependant cette taille dépend du choix du paramètre $x$. En effet, notons $N$ la variable aléatoire prenant pour valeur la taille des objets générés par $\Gamma[\mathcal{A}](x)$, alors $\mathbb{P}_x(N = n) = \frac{a_n x^n}{A(x)}$. Ceci implique,

$$\begin{aligned}
\mathbb{E}(N) &= \sum_{n=0}^{\infty} n\mathbb{P}_x(N = n) \\
&= \frac{x}{A(x)}\left(\sum_{n=0}^{\infty} n\, a_n x^{n-1}\right) \\
&= \frac{xA'(x)}{A(x)}.
\end{aligned}$$

On en déduit en utilisant la même idée que $E(N^2) = \frac{x^2A(x)'' + xA'(x)}{A(x)}$. La variance est alors donnée par :

$$\mathbb{V}ar(N) = \mathbb{E}(N^2) - \mathbb{E}(N)^2 = \frac{A(x)(x^2A(x)'' + xA'(x)) - x^2A(x)^2}{A(x)'^2}.$$

De la même manière qu'une classe combinatoire peut être construite à partir d'autres classes plus simples, les générateurs de Boltzmann peuvent eux aussi faire appel à d'autres générateurs. Pour prouver qu'un algorithme est un générateur de Boltzmann pour une classe $(\mathcal{A}, |.|)$, il suffit de montrer que pour $\alpha \in \mathcal{A}$ et $x \in ]0, R_A[$, on a

$$\mathbb{P}_x(\alpha) = \frac{x^{|\alpha|}}{A(x)}.$$

Comme pour les classes combinatoires, on peut utiliser la méthode symbolique pour construire un générateur de Botzmann récursivement à partir d'autres générateurs plus simples.

### Union disjointe

Notons $\mathcal{B}er(p)$ un algorithme qui retourne une variable aléatoire distribuée selon une loi de Bernoulli de paramètre $p$. Il retourne donc $1$ avec probabilité $p$ et $0$ sinon.

**Algorithme 18:** $\Gamma_+[\mathcal{B}, \mathcal{C}](x)$

```
si Ber(B(x)/(B(x)+C(x))) = 1 alors
    renvoyer Γ[B](x) ;
sinon
    renvoyer Γ[C](x) ;
```

**Proposition 2.** *L'algorithme 18 est un générateur de Boltzmann de la classe $\mathcal{A} = \mathcal{B} + \mathcal{C}$.*

*Démonstration.* Comme $\mathcal{A} = \mathcal{B} + \mathcal{C}$, on a $A(x) = B(x) + C(x)$. Pour $\alpha \in \mathcal{B}$, la probabilité pour l'algorithme 18 de renvoyer $\alpha$ est

$$\mathbb{P}_x(\alpha) = \mathbb{P}(\mathcal{B}er(\frac{B(x)}{B(x)+C(x)}) = 1)\mathbb{P}(\Gamma[\mathcal{B}](x) = \alpha) = \frac{B(x)}{B(x)+C(x)}\frac{x^{|\alpha|_\mathcal{B}}}{B(x)} = \frac{x^{|\alpha|_\mathcal{A}}}{A(x)}.$$

Symétriquement, pour $\alpha \in \mathcal{C}$, la probabilité pour l'algorithme 18 de renvoyer $\alpha$ est

$$\mathbb{P}_x(\alpha) = \left(1 - \frac{B(x)}{B(x)+C(x)}\right)\frac{x^{|\alpha|_\mathcal{C}}}{C(x)} = \frac{C(x)}{A(x)}\frac{x^{|\alpha|_\mathcal{C}}}{C(x)} = \frac{x^{|\alpha|_\mathcal{A}}}{A(x)}.$$

Ainsi l'algorithme 18 est bien un générateur de Boltzmann pour $\mathcal{A} = \mathcal{B} + \mathcal{C}$. □

### Produit cartésien

**Algorithme 19:** $\Gamma_\times[\mathcal{B}, \mathcal{C}](x)$.

```
renvoyer (Γ[B](x), Γ[C](x))
```

**Proposition 3.** *L'algorithme 19 est un générateur de Boltzmann de la classe $\mathcal{A} = \mathcal{B} \times \mathcal{C}$.*

*Démonstration.* Comme $\mathcal{A} = \mathcal{B} \times \mathcal{C}$, $A(x) = B(x)C(x)$. Pour $\alpha \in \mathcal{A}$ il existe $\beta$ et $\gamma$ tels que $\alpha = (\beta, \gamma)$. Ainsi,

$$\mathbb{P}_x(\alpha) = \mathbb{P}(\Gamma[\mathcal{B}](x) = \beta)\mathbb{P}(\Gamma[\mathcal{C}](x) = \gamma) = \frac{x^{|\beta|_\mathcal{B}}x^{|\gamma|_\mathcal{C}}}{B(x)C(x)} = \frac{x^{|\alpha|_\mathcal{A}}}{A(x)}.$$

L'algorithme 19 est bien un générateur de Boltzmann pour $\mathcal{A} = \mathcal{B} \times \mathcal{C}$. □

### Séquence

Soit $\mathcal{G}eo(p)$ un algorithme qui retourne une variable aléatoire distribuée selon une loi géométrique de paramètre $p$. Ce dernier retourne $k \in \mathbb{N}^*$ avec probabilité $p(1-p)^{k-1}$.

**Algorithme 20:** $\Gamma_{\text{SEQ}}[\mathcal{B}](x)$

```
1 k ← Geo(1 − B(x)) ;
2 renvoyer le (k − 1)-uplet (Γ[B](x), ..., Γ[B](x))
```

Justifions l'algorithme 20. La spécification $\mathcal{A} = \mathcal{E} + \mathcal{B} \times \mathcal{A}$ implique qu'on peut utiliser le générateur $\Gamma_+[\mathcal{E}, \mathcal{B} \times \mathcal{A}]$ récursivement jusqu'à la génération de l'élément neutre. À chaque appel de $\Gamma_+[\mathcal{E}, \mathcal{B} \times \mathcal{A}]$, la probabilité de s'arrêter est $\frac{1}{A(x)} = 1 - B(x)$ et celle de continuer $\frac{B(x)A(x)}{A(x)} = B(x)$. Si on note $K$ la variable aléatoire qui compte le nombre d'appels à $\Gamma_+[\mathcal{E}, \mathcal{B} \times \mathcal{A}]$, alors $K$ suit une loi géométrique de paramètre $1 - B(x)$. L'algorithme 20 choisit donc $k$ aléatoirement suivant la loi $\mathcal{G}eo(1 - B(x))$ puis génère $(k-1)$ fois $\beta \in \mathcal{B}$ en faisant appel à $\Gamma[\mathcal{B}]$.

**Proposition 4.** *L'algorithme 20 est un générateur de Boltzmann de la classe* $\mathcal{A} = \text{SEQ}(\mathcal{B})$.

*Démonstration.* Comme $\mathcal{A} = \text{SEQ}(\mathcal{B})$, $A(x) = 1 + B(x)A(x)$ et $1 - B(x) = \frac{1}{A(x)}$. Considérons $\alpha = (\beta_1, \ldots, \beta_k)$. Alors,

$$\mathbb{P}_x(\alpha) = B(x)^{k-1}(1 - B(x)) \prod_{i=1}^{k} \frac{x^{|\beta_k|}}{B(x)} = \frac{B(x)^k}{A(x)} \frac{x^{|\beta_1|+\ldots+|\beta_k|}}{B(x)^k} = \frac{x^{|\alpha|}}{A(x)}.$$

□

**Exemple 29** (Mots finis sur un alphabet binaire)**.** *Soit classe $\mathcal{A}$ la classe des mots finis sur un alphabet binaire, on a vu que $\mathcal{A}$ pouvait s'écrire comme la séquence de l'union des classes atomiques $\mathcal{B}$ et $\mathcal{C}$. L'algorithme 21 est un générateur de Boltzmann pour $\mathcal{A}$, il utilise les générateurs pour les séquences ($\mathcal{A} = \text{SEQ}(\mathcal{B} + \mathcal{C})$), l'union disjointe ($\mathcal{B} + \mathcal{C}$) et les classes atomiques ( $\mathcal{C}$ et $\mathcal{B}$).*

| | |
|---|---|
| 1 | 11100010011010001110100100011110010001010110011010111010 |
| 2 | 01110001110010001101110000010011110110101001100001001 |
| 3 | 010001000101110011010010000111001110101010001000111101100111111001100 |
| 4 | 00100110101010000101111100101011 |
| 5 | 10001100000000011110001010000110111100001011101111100011010111110111011 |

*Les mots binaires ci-dessus ont été obtenus avec en lançant 5 fois un programme correspondant à l'algorithme 21 pour $x = 0.25$.*

## 6.2 Échantillonnage de Boltzmann des réseaux de Gordon et Newell

Nous avons vu dans la section 1.2 que la distribution stationnaire d'un réseau de Gordon et Newell possède la forme produit et donc peut directement être exprimée en fonction des paramètres du réseau.

**Algorithme 21:** Générateur de Boltzmann des mots sur $\{0,1\}^*$.

```
k ← Geo(1 − 2x) − 1 ;
w ← "" ;
pour i = {1, 2, ..., k} faire
    si Ber(p) = 1 alors
        w ← w + "0" ;
    sinon
        w ← w + "1" ;
renvoyer w
```

**Théorème 14** (Forme produit d'un réseau de Gordon et Newell)**.** *La probabilité stationnaire d'un réseau de Gordon et Newell composé de $K$ files $./M/1/\infty/1/FIFO$ et de $M$ clients est telle que*

$$\mathbb{P}(\mathbf{x}) = \frac{1}{G(K,M)} \prod_{k=1}^{K} \rho_k^{x_k} \quad \textit{avec } \rho_k = \frac{h_k}{\mu_k},$$

$$G(K,M) = \sum_{\mathbf{x}\in\mathcal{S}} \prod_{k\in\mathcal{Q}} \rho_k^{x_k} \quad \textit{et} \quad \mathcal{S} = \{\mathbf{x} = (x_1, x_2, \ldots, x_K) \in \mathbb{N}^K \ : \ \sum_{k=1}^{K} x_k = M\}.$$

Au vu de la section précédente, l'espace des états $\mathcal{S}$ s'apparente à une classe combinatoire et la constante de normalisation $G(K,M)$ à une fonction génératrice. On s'intéresse dans cette section à l'aspect combinatoire de l'espace des états d'un réseau fermé de file d'attente. On proposera une classe combinatoire et une fonction génératrice associée à l'espace des états. Cette fonction sera multivariée. On présentera ensuite deux générateurs de Boltzmann capables d'échantillonner la distribution stationnaire d'un réseaux de Gordon et Newel. Dans [8], Bodini et Ponty proposent la première extension des générateurs de Boltzmann pour une classe combinatoire multivariée.

### 6.2.1 Classe combinatoire et fonction génératrice

On note $(\mathcal{A}_K, |.|)$ la classe combinatoire telle que

$$\mathcal{A}_K = \{\alpha = (\alpha_1, \ldots, \alpha_K) \in \mathbb{N}^K\},$$

$$\begin{array}{rccc} |.|_0 \ : & \mathcal{A}_K & \to & \mathbb{N} \\ & \alpha & \mapsto & \sum_{k=1}^{K} \alpha_k, \end{array}$$

et pour tout $k \in \{1, \ldots, K\}$,

$$\begin{array}{rccc} |.|_k \ : & \mathcal{A}_K & \to & \mathbb{N} \\ & \alpha & \mapsto & \alpha_k. \end{array}$$

Soit $\alpha \in \mathcal{A}_K$, alors $\alpha$ peut être vu comme un état d'un réseau de $K$ files d'attente. La fonction taille $|.|_0$ compte le nombre de clients au total et les $|.|_k$ le nombre de clients dans chaque file $k \in \mathcal{Q}$.

Pour $M \in \mathbb{N}$ fixé, on note

$$\mathcal{A}_{K,M} = \left\{\alpha = (\alpha_1, \ldots, \alpha_K) \in \mathbb{N}^K \mid |\alpha|_0 = M\right\}.$$

L'ensemble $\mathcal{A}_{K,M}$ est isomorphe à l'espace des états $\mathcal{S}$ du théorème 14 et on a donc $a_{K,M} = \binom{K+M-1}{K-1}$. Notons $A_K$ la fonction génératrice de la classe combinatoire multivariée $(\mathcal{A}_K, |.|)$ et $\mathbf{u}$ un vecteur de taille $K$, alors

$$\begin{aligned} A_K(z,\mathbf{u}) &= \sum_{\alpha\in\mathcal{A}_K} z^{|\alpha|_0} {u_1}^{|\alpha|_1}\dots{u_K}^{|\alpha|_K} && (6.6)\\ &= \sum_{M\geq 0}\left(\sum_{\alpha\in\mathcal{A}_{K,M}} z^{\alpha_1+\dots+\alpha_K}{u_1}^{\alpha_1}\dots{u_K}^{\alpha_K}\right) && (6.7)\\ &= \prod_{i=1}^{K}\left(\sum_{m_i=0}^{\infty} {u_i}^{m_i} z^{m_i}\right) && (6.8)\\ &= \prod_{i=1}^{K}\frac{1}{1-u_i z}. && (6.9) \end{aligned}$$

Du point de vue des files d'attente, la variable $z$ marque le nombre de clients total dans le réseau et les $u_k$ marquent le nombre de clients présents en file $k$.

### 6.2.2 Générateur de Boltzmann

Maintenant que nous avons identifié la classe combinatoire et la fonction génératrice, on cherche à construire un générateur de Boltzmann capable d'échantillonner la distribution stationnaire du théorème 14. Les paramètres du générateur $x$ et $\rho = (\rho_1,\dots,\rho_K)$ doivent être dans le rayon de convergence de $A_K(x,\rho)$, c'est-à-dire tels que pour tout $k\in\{1,2,\dots,K\}$ $0 < x\rho_k < 1$. On commence par fixer $\rho = (\rho_1,\dots,\rho_K)$ selon le théorème 14.

Un générateur de Boltzmann de paramètres $(x,\rho)$ de la classe $\mathcal{A}_K$ est un algorithme probabiliste qui produit $\alpha\in\mathcal{A}_K$ avec probabilité $\mathbb{P}_{x,\rho}(\alpha)$ telle que,

$$\mathbb{P}_{x,\rho}(\alpha) = \frac{x^{|\alpha|_0}\rho_1^{|\alpha_1|_1}\cdots\rho_K^{|\alpha_K|_K}}{A_K(x,\rho)} = \frac{x^{\sum_{k=1}^{K}\alpha_k}\rho_1^{\alpha_1}\cdots\rho_K^{\alpha_K}}{A_K(x,\rho)}. \quad (6.10)$$

Pour $M\in\mathbb{N}$, on a

$$\begin{aligned} \mathbb{P}_{x,\rho}(\alpha||\alpha|_0 = M) &= \frac{\mathbb{P}_{x,\rho}(\alpha,|\alpha|_0=M)}{\mathbb{P}_{x,\rho}(|\alpha|_0=M)}\\ &= \frac{x^M\rho_1^{\alpha_1}\cdots\rho_K^{\alpha_K}}{A_K(x,\rho)\mathbb{P}_{x,\rho}(|\alpha|_0=M)} \end{aligned}$$

Or

$$\mathbb{P}_{x,\rho}(|\alpha|_0=M) = \frac{x^M}{A_K(x,\rho)}\sum_{\alpha\in\mathcal{A}_{K,M}}\left(\prod_{i=1}^{K}\rho_i^{\alpha_i}\right). \quad (6.11)$$

En faisant le lien avec le théorème 14 on obtient

$$\begin{aligned}\mathbb{P}_{x,\rho}(|\alpha|_0 = M) &= \frac{x^M}{A_K(x,\rho)} \sum_{\mathbf{y}\in\mathcal{S}} \left( \prod_{i\in\mathcal{Q}} \rho_i^{y_i} \right) \\ &= \frac{x^M G(K,M)}{A_K(x,\rho)}. \end{aligned} \tag{6.12}$$

$$\mathbb{P}_{x,\rho}(\alpha||\alpha|_0 = M) = \frac{x^M \rho_1^{\alpha_1} \cdots \rho_K^{\alpha_K} A_K(x,\rho)}{x^M A_K(x,\rho) G(M,K)} = \frac{1}{G(K,M)} \prod_{k=1}^{K} {\rho_k}^{\alpha_k}. \tag{6.13}$$

La formule (6.13) correspond exactement à la probabilité de générer un état selon la distribution stationnaire du théorème 14. Ainsi a méthode de Boltzmann peut être utilisée pour générer des échantillons distribués selon la distribution stationnaire d'un réseau de Gordon et Newel. Cependant il générera des échantillons pour lesquels le nombre de clients $M$ est non fixé. Le paramètre $\rho$ est donné par le théorème 14 tandis que le paramètre $x$ sert à contrôler $M$ le nombre de clients. Pour obtenir des échantillons avec un nombre de clients fixé, on pourra comme c'est le cas des générateurs de Boltzmann en général, effectuer du rejet. À partir de la formulation (6.9) de $\mathcal{A}_K(z,\mathbf{u})$ on propose deux générateurs de Boltzmann.

### Première proposition

La formulation (6.9) implique que pour tout $k \geq 2$,

$$A_k(z,\mathbf{u}) = \frac{A_{k-1}(z,\mathbf{u})}{1-u_k z}.$$

Ainsi pour $x$ et $\rho$ fixés on a,

$$\begin{cases} A_0(x,\rho) = 1 \\ A_k(x,\rho) = A_{k-1}(x,\rho) + \rho_k x A_k(x,\rho) \text{ pour tout } k \geq 2 \end{cases}$$

Notons $\mathbf{e_K}$ le vecteur de taille $K$ tel que $\mathbf{e_K} = 1$ et $\mathbf{e_k} = 0$ pour tout $k < K$.

**Algorithme 22:** $\Gamma_{GN1}[K](x,\rho)$

```
si K = 0 alors
    renvoyer ∅;
sinon
    si Ber( A_{K-1}(x,ρ) / A_K(x,ρ) ) = 1 alors
        (Γ_{GN1}[K-1](x,ρ), 0);
    sinon
        Γ_{GN1}[K](x,ρ) + e_K;
```

**Lemme 25.** *L'algorithme 22 est un générateur de Boltzmann de la classe $\mathcal{A}_K$.*

*Démonstration.* On procède par récurrence sur $k \in \{1, 2, \ldots, K\}$.

Considérons le cas $k = 0$, alors $A_k(x, \rho) = 1$ et $\mathcal{A}_k = \mathcal{E}$ la classe neutre. Le générateur $\Gamma_{GN1}[0](x, \rho) = \Gamma_{\text{SEQ}}[\mathcal{Z}](x\ \rho_1)$ est bien un générateur de Boltzmann pour la classe neutre.

Supposons maintenant $k > 0$ et que $\Gamma_{GN1}[k](x, \rho)$ soit un générateur de Boltzmann de la classe $\mathcal{A}_k$. On a alors

$$\mathbb{P}_{k,x,\rho}(\alpha) = \frac{x^{\sum_{i=1}^{k} \alpha_i} \rho_1^{\alpha_1} \cdots \rho_k^{\alpha_k}}{A_k(x, \rho)}.$$

Soit $\alpha \in \mathcal{A}_{k+1}$ montrons que $\Gamma_{GN1}[k + 1](x, \rho)$ est un générateur de Boltzmann de la classe $\mathcal{A}_{k+1}$. On distingue deux cas : $\alpha_{k+1} = 0$ et $\alpha_{k+1} > 0$.

**Cas 1 :** $\alpha_{k+1} = 0$

$$\begin{aligned}
\mathbb{P}_{k+1,x,\rho}(\alpha) &= \mathbb{P}\left(\mathcal{B}er\left(\frac{A_k(x, \rho)}{A_{k+1}(x, \rho)}\right) = 1\right) \mathbb{P}\left(\Gamma_{BN1}[k](x, \rho) = (\alpha_1, \ldots, \alpha_k)\right) \\
&= \frac{A_k(x, \rho)}{A_{k+1}(x, \rho)} \mathbb{P}_{k,x,\rho}(\alpha) \\
&= \frac{x^{\sum_{i=1}^{k+1} \alpha_i} \rho_1^{\alpha_1} \cdots \rho_{k+1}^{\alpha_{k+1}}}{A_{k+1}(x, \rho)}
\end{aligned}$$

**Cas 2 :** $\alpha_{k+1} > 0$

$$\begin{aligned}
\mathbb{P}_{k+1,x,\rho}(\alpha) &= \mathbb{P}\left(\mathcal{B}er\left(\frac{A_k(x, \rho)}{A_{k+1}(x, \rho)}\right) = 0\right)^{\alpha_{k+1}} \mathbb{P}\left(\mathcal{B}er\left(\frac{A_k(x, \rho)}{A_{k+1}(x, \rho)}\right) = 1\right) \mathbb{P}(\Gamma_{BN1}[k](x, \rho) = (\alpha_1, \ldots, \alpha_k)) \\
&= \left(\frac{x\rho_{k+1} A_{k+1}(x, \rho)}{A_{k+1}(x, \rho)}\right)^{\alpha_{k+1}} \frac{A_k(x, \rho)}{A_{k+1}(x, \rho)} \mathbb{P}_{k,x,\rho}(\alpha) \\
&= \frac{x^{\sum_{i=1}^{k+1} \alpha_i} \rho_1^{\alpha_1} \cdots \rho_{k+1}^{\alpha_{k+1}}}{A_{k+1}(x, \rho)}
\end{aligned}$$

La formule (6.10) est vérifiée, l'algorithme 22 est bien un générateur de Boltzmann. □

Remarquons que ce générateur utilise la même idée que Buzen [19] pour le calcul de la constante de normalisation $G(K, M)$.

### Seconde proposition

On utilise directement la formule (6.9),

$$A(z, \mathbf{u}) = \prod_{i=1}^{K} \left(\frac{1}{1 - u_i z}\right).$$

Soit $\mathcal{Z}$ la classe atomique, on a alors $Z(z) = z$. La fonction génératrice de la classe $\mathcal{B} = \text{SEQ}(\mathcal{Z})$ est égale à $B(z) = \frac{1}{1-Z(z)} = \frac{1}{1-z}$. Ainsi on obtient

$$A(z, \mathbf{u}) = \prod_{i=1}^{K} B(u_k z).$$

**Algorithme 23:** $\Gamma_{BN2}[K](x, \rho)$.

1 **renvoyer** $(\Gamma_{\text{SEQ}}[\mathcal{Z}](x\rho_1), \ldots, \Gamma_{\text{SEQ}}[\mathcal{Z}](x\rho_K))$ ;

**Lemme 26.** *L'algorithme $\Gamma_{BN2}[K](x, \rho)$ est un générateur de Boltzmann de la classe $\mathcal{A}_K$.*

*Démonstration.* Considérons $\alpha = (\alpha_1, \ldots, \alpha_K)$ alors la probabilité pour l'algorithme 23 de retourner $\alpha$ est égal au produit des probabilités des algorithmes $\Gamma_{\text{SEQ}}[\mathcal{Z}](x\rho_k)$ de retourner $\alpha_k$, on en déduit alors

$$\mathbb{P}_{x,\rho}(\alpha) = \prod_{k=1}^{K} \mathbb{P}_{\Gamma_{\text{SEQ}}[\mathcal{Z}](x\rho_k)}(\alpha_k) = \prod_{k=1}^{K} \frac{(x\ \rho_k)^{\alpha_k}}{1 - (x\rho_k)} = \frac{x^{\sum_{k=1}^{K} \alpha_k}\ \rho_1^{\alpha_1} \cdots \rho_K^{\alpha_K}}{A_K(x, \rho)}.$$

La formule (6.10) est ainsi vérifiée.

□

**Remarque 1.** *Dans la section 2.2.6 nous avons montré qu'en appliquant l'algorithme* `RandState` *au diagramme complet $\mathcal{D}(K, M)$ des sommes cumulées et en choisissant bien le poids de ses arcs,* `RandState` *est un algorithme probabiliste qui génère des états distribués selon la distribution stationnaire du théorème 14. Ainsi, cette utilisation de* `RandState` *produit un générateur de Boltzmann de paramètre $(\rho_1, \ldots, \rho_K)$ pour la classe combinatoire $\mathcal{A}_{K,M}$.*

# Conclusion

Dans ce chapitre on a rappelé les principaux concepts de la génération de Boltzmann. On en ensuite montré que l'espace des états d'un réseau de Gordon et Newell possédant $K$ files et $M$ clients peut être représenté par une classe combinatoire. Dans cette classe, la fonction taille est de dimension $K + 1$ et compte le nombre total de clients ainsi que celui dans chaque file. À partir de la spécification de cette classe, on a construit deux générateurs de Boltzmann. Ces derniers échantillonnent la distribution stationnaire d'un réseau de Gordon Newell pour un nombre de clients total $M$ non fixé. Ce résultat semble être généralisable aux réseaux multi-classes BCMP [4] car comme les réseaux de Gordon Newell, leur distribution stationnaire est à forme produit. Ceci pourra faire l'objet de future recherches.

Dans le chapitre suivant on s'intéressera aux générateurs de Boltzmann pour la classe combinatoire des multi-ensembles. On présentera un générateur de Boltzmann basé sur un diagramme complet pour les multi-ensembles de cardinalité fixe.

# 7 Échantillonnage de Boltzmann pour les multi-ensembles

Ce chapitre a pour objet les générateurs de Boltzmann pour les multi-ensembles et notamment ceux de cardinalité fixe qui furent introduit par Flajolet *et al.* dans [26]. La première section présente les résultats de [26] concernant les multi-ensembles. Dans la seconde section on s'intéressera aux partitions d'entiers, on montrera notamment que tout ensemble de partitions peut être encodé par un diagramme et on proposera un autre générateur de Boltzmann basé sur les diagrammes pour la génération des multi-ensembles de cardinalité fixe.

## 7.1 Générateur de Boltzmann pour les multi-ensembles

### 7.1.1 Diagonale d'une classe combinatoire

Avant de commencer à étudier les multi-ensembles nous avons besoin d'introduire la notion de diagonale. On note $\beta^{\odot k} = \langle \beta, \dots, \beta \rangle$ le $k$-uplet contenant $k$ fois l'élément $\beta \in \mathcal{B}$. La **diagonale** à $k$ éléments de la classe combinatoire de structure $\mathcal{B}$ est définie par

$$\Delta_k \mathcal{B} := \{\beta^{\odot k} | \beta \in \mathcal{B}\}.$$

La fonction génératrice associée à $\mathcal{A} = \Delta_k \mathcal{B}$ est alors $A(z) = B(z^k)$. Par convention, on pose $\alpha^{\odot 0} := \mathcal{E}$ (la classe neutre).

**Algorithme 24:** $\Gamma_{\Delta_k}[\mathcal{B}](x)$.

1 $\beta \leftarrow \Gamma[\mathcal{B}](x^k)$ ;
2 **renvoyer** *le $k$-uplet* $\beta^{\odot k}$

La probabilité pour $\Gamma_{\Delta_k}[\mathcal{B}](x)$ d'engendrer le $k$-uplet $\gamma = \beta^{\odot k}$ est donnée par

$$\mathbb{P}_x(\gamma) = \frac{x^{k|\beta|}}{B(x^k)} = \frac{x^{|\gamma|}}{A(x)},$$

ce qui en fait bien un générateur valide pour la classe $\mathcal{A} = \Delta_k \mathcal{B}$.

### 7.1.2 Multi-ensembles

Un multi-ensemble est une généralisation de la notion d'ensemble dans lequel chaque élément peut apparaître un nombre fini de fois. On note $\mathcal{A} = \textsc{mset}(\mathcal{B})$ le multi-ensemble obtenu à partir des éléments de $\mathcal{B}$.

**Exemple 30.** *Soit $\mathcal{B}$ un ensemble représenté en figure 7.1. Alors l'élément $\alpha$ représenté en figure 7.2 est tel que $\alpha \in \textsc{mset}(\mathcal{B})$. Dans $\alpha$, le cercle et le carré sont de multiplicité 2, le triangle de multiplicité 1 et les autres éléments de $\mathcal{B}$ sont multiplicité 0.*

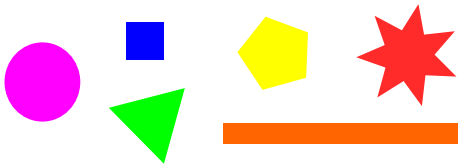


**FIGURE 7.1:** Ensemble $\mathcal{B}$.

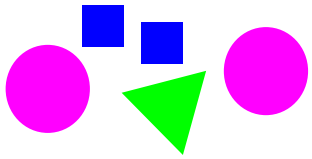


**FIGURE 7.2:** $\alpha \in \textsc{mset}(\mathcal{B})$.

Le multi-ensemble $\mathcal{A} = \textsc{mset}(\mathcal{B})$ peut être défini comme le produit cartésien des séquences issues de chaque élément de $\mathcal{B}$ (voir l'exemple 31).

$$\mathcal{A} = \textsc{mset}(\mathcal{B}) \tilde{=} \prod_{\beta \in \mathcal{B}} \textsc{seq}(\beta). \tag{7.1}$$

**Exemple 31.** *Soit* $\mathcal{B} = \{\beta, \beta'\}$,

$$\begin{aligned}
\textsc{seq}(\{\beta\})\textsc{seq}(\{\beta'\}) &= \sum_{k=0}^{\infty} \beta^{\odot k} \sum_{l=0}^{\infty} \beta'^{\odot l} \\
&= \mathcal{E} + \langle\beta\rangle + \langle\beta'\rangle + \langle\beta\rangle\langle\beta\rangle + \langle\beta\rangle\langle\beta'\rangle + \dots \\
&\simeq \mathcal{E} + \langle\beta\rangle + \langle\beta'\rangle + \langle\beta, \beta\rangle + \langle\beta, \beta'\rangle + \langle\beta', \beta'\rangle + \dots \\
&= \textsc{mset}(\mathcal{B})
\end{aligned}$$

À partir de la formule (7.1), on déduit $A(z)$ la fonction génératrice pour les multi-ensembles :

$$A(z) = \prod_{\beta \in \mathcal{B}} (1 + z^{|\beta|} + z^{2|\beta|+\cdots}) = \prod_{\beta \in \mathcal{B}} \frac{1}{1 - z^{|\beta|}}. \tag{7.2}$$

Un générateur de Boltzmann pour les des multi-ensembles se déduit facilement de la spécification (7.1) en parcourant tous les élément de l'ensemble $\mathcal{B}$. On commence par initialiser le multi-ensemble que l'on veut générer à l'ensemble vide puis on génère pour chaque $\beta \in \mathcal{B}$ un élément de $\textsc{seq}(\mathcal{B})$ que l'on ajoute au multi-ensemble. L'algorithme se termine quand on a fini le parcours de tous les éléments de $\mathcal{B}$ et il renvoie le multi-ensemble. Hélas si $\mathcal{B}$ contient un nombre infini d'éléments ce générateur n'est pas utilisable. Dans [26] les auteurs proposent un générateur effectif pour les multi-ensembles. Pour chaque génération de $\alpha \in \textsc{mset}(\mathcal{B})$, ils commencent par fixer un indice $k \in \mathbb{N}$ pour ensuite ne générer que des éléments $\beta^{\odot j}$ tels que $j \leq k$.

**Exemple 32.** *Considérons $\mathcal{B}$ l'ensemble représenté en figure 7.1. La figure 7.3 illustre un élément $\beta \in \textsc{mset}(\mathcal{B})$. Pour construire $\beta \in \textsc{mset}(\mathcal{B})$, on peut choisir pour indice de multiplicité maximal $k = 3$ et pour chaque multiplicité $j \in \{1, 2, 3\}$ prendre pour :*

- *$j = 1$ : un cercle, un triangle, un carré et un rectangle ;*
- *$j = 2$ : deux triangles et deux pentagones ;*
- *$j = 3$ : trois carrés.*

*Ainsi on obtient bien le multi-ensemble $\beta \in \textsc{mset}(\mathcal{B})$ qui est composé de 4 carrés, 3 triangles, 2 pentagones, 1 cercle et 1 rectangle.*

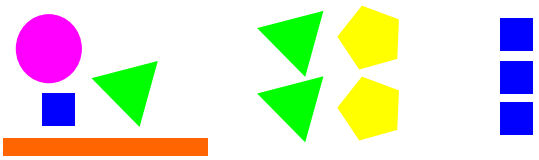


**FIGURE 7.3:** $\beta \in \textsc{mset}(\mathcal{B})$.

Pour construire le générateur effectif, une autre formulation de $A(z)$ est donnée en utilisant $f = \exp(\ln(f))$ et le développement en série de Maclaurin du logarithme.

$$\begin{aligned}
A(z) &= \prod_{\beta\in\mathcal{B}} \frac{1}{1-z^{|\beta|}} \\
&= \prod_{n=1}^{\infty}(1-z^n)^{-b_n} \\
&= \exp\left(\sum_{n=1}^{\infty} -b_n \ln(1-z^n)\right) \\
&= \exp\left(\sum_{n=1}^{\infty} -b_n \left(\sum_{k=1}^{\infty} \frac{-z^{nk}}{k}\right)\right) \\
&= \exp\left(\sum_{k=1}^{\infty} \frac{1}{k}(\sum_{n=1}^{\infty} b_n z^{k^n})\right) \\
&= \exp\left(\sum_{k=1}^{\infty} \frac{B(z^k)}{k}\right) \\
&= \prod_{k=1}^{\infty} \exp\left(\frac{B(z^k)}{k}\right).
\end{aligned}$$

On en déduit donc une nouvelle équation pour la fonction génératrice des multi-ensembles :

$$A(z) = \prod_{j=1}^{\infty} \exp\big(\frac{B(z^j)}{j}\big). \tag{7.3}$$

À partir de l'équation (7.3) on déduit le générateur de Boltzmann pour les multi-ensembles (cf. algorithme 25). On considère les multi-ensembles comme le produit de structures ayant pour fonction génératrice $\exp\big(\frac{B(z^j)}{j}\big)$. La contrainte sur la limite ($k = \infty$) est éliminée en considérant la variable aléatoire $K$ qui prend pour valeur l'indice maximal des diagonales de l'élément à générer. La distribution de $K$ est telle que

$$\mathbb{P}_x(K = k) = \frac{\prod_{j=1}^{k} \exp\big(\frac{B(z^j)}{j}\big)}{A(x)}.$$

Considérons $IndiceMax(\mathcal{B}, x)$ un algorithme qui renvoie $k$ selon la distribution $\mathbb{P}_x(K = k)$. L'algorithme 25 commence par déterminer $k$ avec probabilité $\mathbb{P}_x(K = k)$ puis génère le multi-ensemble diagonale par diagonale.

Notations :

- $Poiss(\lambda)$ renvoie une variable aléatoire distribuée selon une loi de Poisson de paramètre $\lambda$ ;
- $Poiss_{\geq 1}(\lambda)$ renvoie une variable aléatoire strictement positive distribuée selon une loi de Poisson de paramètre $\lambda$ ;
- $\biguplus$ est l'opération additive pour les multi-ensembles, par exemple $\langle \alpha, \beta\rangle \biguplus \langle\beta\rangle = \langle \alpha, \beta^{\odot 2}\rangle$.

**Proposition 5** (Flajolet et al. [26])**.** $\Gamma_{\text{MSET}}[\mathcal{B}](x)$ *est un générateur de Boltzmann pour la classe* $\text{MSET}(\mathcal{B})$.

**Algorithme 25:** $\Gamma_{\text{MSET}}[\mathcal{B}](x)$

$\alpha \leftarrow \emptyset$;
$k \leftarrow IndiceMax(\mathcal{B}, x)$;
**si** $k > 0$ **alors**
  **pour** $j = 1, 2, \ldots, k-1$ **faire**
    $n_j \leftarrow Poiss(\frac{B(x^j)}{j})$;
    **pour** $i = 1, \ldots, n_j$ **faire**
      $\gamma \leftarrow \Gamma_{\Delta_j}[\mathcal{B}](x)$;
      $\alpha \leftarrow \alpha \uplus \{\gamma\}$;
    $n_k \leftarrow Poiss_{\geq 1}(\frac{B(x^k)}{k})$;
    **pour** $i = 1, \ldots, n_k$ **faire**
      $\gamma \leftarrow \Gamma_{\Delta_i}[\mathcal{B}](x)$;
      $\alpha \leftarrow \alpha \uplus \{\gamma\}$;
**renvoyer** $\alpha$

### 7.1.3 Multi-ensembles de cardinalité fixe

On appelle **multi-ensemble à $K$ éléments** un un multi-ensemble contenant exactement $K$ éléments. La classe de tels ensembles est notée $\text{MSET}_K(\mathcal{B})$ et est définie par :

$$\text{MSET}_K(\mathcal{B}) = \left\{ \langle \beta_1^{\odot m_1}, \beta_2^{\odot m_2}, \ldots \rangle \ \in \text{MSET}(\mathcal{B}) \mid \sum_{i \geq 0} m_i = K \right\}.$$

Dans les exemples précédents, on a $\alpha \in \text{MSET}_5(\mathcal{B})$ (voir figure 7.2) et $\beta \in \text{MSET}_{11}(\mathcal{B})$ (voir figure 7.3). On note $A_K(z)$ la fonction génératrice associée à $\text{MSET}_K(\mathcal{B})$. Pour déterminer $A_K(z)$ ([26]) on se sert de l'inclusion $\text{MSET}_K(\mathcal{B}) \subset \text{MSET}(\mathcal{B})$ on extrait $A_K(z)$ à partir de $A(z)$ (la fonction génératrice associée à $\text{MSET}(\mathcal{B})$). Pour ce faire, on considère une nouvelle fonction génératrice associée elle aussi à $\text{MSET}(\mathcal{B})$ mais qui dispose de deux variables. Notons la $A(z, u)$. La variable $u$ sert a marquer le nombre d'éléments présents dans le multi-ensemble. On a alors,

$$A(z) = \prod_{\beta \in \mathcal{B}} \frac{1}{1 - z^{|\beta|}} \quad \text{et} \quad A(z, u) = \prod_{\beta \in \mathcal{B}} \frac{1}{1 - u z^{|\beta|}}.$$

**Exemple 33.** *Pour $\mathcal{B} = \{\beta, \beta'\}$ et $\mathcal{A} = \text{MSET}(\mathcal{B})$, on a $A(z) = \frac{1}{1-z^{|\beta|}} \times \frac{1}{1-z^{|\beta'|}}$ et*

$$\begin{aligned}
A(z,u) &= \frac{1}{1-uz^{\beta}} \times \frac{1}{1-uz^{\beta'}} \\
&= (1 + uz^{|\beta|} + u^2 z^{2|\beta|} + \ldots)(1 + uz^{|\beta'|} + u^2 z^{2|\beta'|} + \ldots) \\
&= 1 + uz^{|\beta|} + uz^{|\beta'|} + u^2 z^{2|\beta|} + u^2 z^{|\beta|+|\beta'|} + u^2 z^{2|\beta'|} + \cdots \\
&= \sum_{k,\ell \geq 0} \left(u\, z^{|\beta|}\right)^k \left(u\, z^{|\beta'|}\right)^\ell \\
&= \sum_{k,\ell \geq 0} u^{k+\ell} z^{k|\beta|+\ell|\beta'|} \\
&= \sum_{k,\ell \geq 0} u^{k+\ell} z^{|\beta^{\odot k}|+|\beta'^{\odot \ell}|}.
\end{aligned}$$

En suivant le même raisonnement qui nous a permis d'aboutir à la formule (7.3), on obtient

$$A(z,u) = \exp\big(\sum_{k=1}^{\infty} \frac{u^k B(z^k)}{k}\big),$$

et comme $\exp(x) = \sum_{n=0}^{\infty} \frac{x^n}{n!}$ on en déduit

$$A(z,u) = \sum_{n=0}^{\infty} \frac{1}{n!}\Big(\sum_{i=1}^{\infty} \frac{u^i B(z^i)}{i}\Big)^n. \tag{7.4}$$

Montrons maintenant que l'on retrouve $A_K(z)$ à partir de $A(z,u)$, en utilisant l'opération d'extraction suivante

$$A_K(z) := [u^K]A(z,u) = [u^K]\sum_{n=0}^{\infty} \frac{1}{n!}\Big(\sum_{i=1}^{\infty} \frac{u^i B(z^i)}{i}\Big)^n. \tag{7.5}$$

En effet, la fonction $A_K(z)$ est polynomiale en $B(z), B(z^2), \ldots, B(z^K)$. Par exemple,

- $A_2(z) = \frac{B(z^2)}{2} + \frac{B(z)^2}{2}$
- $A_3(z) = \frac{B(z^3)}{3} + \frac{B(z)B(z^2)}{2} + \frac{B(z)^3}{6}$.

Prenons le cas $K = 3$. Pour trouver l'expression de $A_3(z)$, on regarde quels sont les coefficients multiplicateurs de $u^3$ dans la formule de $A(z,u)$. Premièrement, il n'est pas nécessaire de considérer les $n$ tels que $n > 3$. Pour $n = 1$, il n'y a qu'un choix possible, prendre $i = 3$ dans la deuxième somme. On obtient ainsi pour facteur de $u^3$ : $\frac{B(z^3)}{3}$. Pour $n = 3$, il n'y a aussi qu'un seul choix possible, prendre $i = 1$ trois fois. On obtient $\frac{u^3 B(z)^3}{3!}$. Pour $n = 2$, il y a 2 choix, prendre $i = 1$ puis $i = 2$ et inversement $i = 2$ puis $i = 1$, ce qui donne alors $\frac{B(z)B(z^2)}{2}$.

Pour généraliser la formule $A_K(z)$ on a besoin de définir les partitions d'entiers.

**Définition 13.** *On appelle partition de l'entier $K \in \mathbb{N}$ tout vecteur $\mathbf{p} = (p_1, p_2, \ldots, p_k) \in \mathbb{N}^k$ tel que*

$$\sum_{i=1}^{k} i p_i = K.$$

*On note $\mathcal{P}_E(K)$ l'ensemble des vecteurs de partition de l'entier $K$.*

En 1918, Hardy et Ramanujan [35] ont démontré que

$$|\mathcal{P}_E(K)| \sim \frac{1}{4M\sqrt{3}} \exp\left(\pi\sqrt{\frac{2M}{3}}\right)$$

ainsi on en déduit ainsi que le nombre de partitions de $K$ est en $O(e^{\sqrt{K}})$.

**Exemple 34.** *L'entier 5 admet 7 partitions :*

$\mathcal{P}_E(5) = \{(5,0,0,0,0), (2,0,1,0,0), (3,1,0,0,0), (1,2,0,0,0), (0,1,1,0,0), (0,0,0,0,1), (1,0,0,1,0).\}$

**Lemme 27** (Flajolet et al. [26])**.** *Soit $\mathcal{B}$ une classe combinatoire et $B(z)$ sa fonction génératrice, alors la fonction génératrice de la classe $\textsc{mset}_K(\mathcal{B})$ est donnée par*

$$A_K(z) = \sum_{\mathbf{p}\in\mathcal{P}_E(K)} A_{\mathbf{p}}(z) \text{ avec } A_{\mathbf{p}} = \prod_{i=1}^{k} \frac{B(z^i)^{p_i}}{p_i! i^{p_i}} \text{ et } \mathbf{p} = (p_1, p_2, \ldots, p_K). \tag{7.6}$$

**Exemple 35.** *Pour $k = 3$, les partitions sont les suivantes :* $(3,0,0),(1,1,0),(0,0,1)$. *On retrouve bien* $A_3(z) = \frac{B(z)^3}{6} + \frac{B(z)B(z^2)}{2} + \frac{B(z^3)}{3}$.

*Démonstration.* On intéresse aux $u^K$ dans $A(z,u)$, ainsi on n'a pas besoin de considérer des sommes au-delà de $K$. Ainsi,

$$A_K(z) = [u^K] \sum_{n=0}^{K} \frac{1}{n!} \Big( \sum_{i=1}^{k} \frac{u^i B(z^i)}{i} \Big)^n.$$

En utilisant la formule du multinôme de Newton

$$(x_1 + x_2 + \cdots + x_k)^n = \sum_{n_1+n_2+\ldots+n_m=n} \binom{n}{n_1, n_2, n_3, \ldots, n_m} x_1^{n_1} x_2^{n_2} x_3^{n_3} \ldots x_m^{n_m}$$

et les coefficients multinômiaux

$$\binom{n}{n_1, n_2, n_3, \ldots, n_m} = \frac{n!}{\prod_{i=1}^{k} n_i!},$$

on en déduit une nouvelle expression pour la fonction génératrice $A_K(z)$ :

$$\begin{aligned} A_K(z) &= [u^K] \sum_{n=0}^{K} \frac{1}{n!} \Big( \sum_{n_1+\ldots+n_K=n} \binom{n}{n_1, n_2, \ldots, n_K} \prod_{i=1}^{K} \frac{u^{i\, n_i} B(z^i)^{n_i}}{i^{n_i}} \Big) \\ &= [u^K] \sum_{n=0}^{K} \Big( \sum_{n_1+\ldots+n_K=n} \big( \prod_{i=1}^{K} \frac{u^{i n_i} B(z^i)^{n_i}}{n_i! i^{n_i}} \big) \Big). \end{aligned}$$

Pour l'extraction il nous faut donc considérer les $K$-uplets $(n_1, \ldots, n_K)$ tels que $\prod_{i=1}^{K} u^{i n_i} = u^K$, c'est-à-dire toutes les partitions de l'entier $K$. Ainsi, on a

$$A_K(z) = \sum_{\mathbf{p} \in \mathcal{P}_E(K)} \prod_{i=1}^{K} \frac{B(z^i)^{n_i}}{p_i! i^{p_i}}. \tag{7.7}$$

□

Le générateur de Boltzmann pour les multi-ensembles de cardinalité $K$ (cf. algorithme 26) se déduit à partir de la formulation (7.7). Il utilise les générateurs de base : union disjointe et produit cartésien.

**Algorithme 26:** $\Gamma_{\text{MSET}_K}[\mathcal{B}](x)$.

$\alpha \leftarrow \emptyset$;
Choisir $\mathbf{p} := (p_1, \ldots, p_K) \in \mathcal{P}_E(K)$ avec probabilité $\frac{A_{\mathbf{p}}(x)}{A_k(x)}$;
**pour** $i = 1, 2, \ldots, K$ **faire**
  **pour** $n = 1, 2, \ldots, p_i$ **faire**
    $\alpha \leftarrow \alpha \uplus \Gamma_{\Delta_i}[\mathcal{B}](x)$;
**renvoyer** $\alpha$.

**Proposition 6** (Flajolet et al. [26])**.** $\Gamma_{\text{MSET}_K}[\mathcal{B}](x)$ *est un générateur de Boltzmann pour la classe* $\text{MSET}_K(\mathcal{B})$.

## 7.2 Diagramme pour la génération de multi-ensembles de cardinalité fixe

Dans cette section on propose un autre générateur de Boltzmann pour $\textsc{mset}_K$ qui utilise le diagramme complet des partitions de l'entier $K$.

### 7.2.1 Diagramme des partitions

Montrons pour commencer que l'ensemble des partitions de $K$ peut être représenté par un diagramme. Une partition de l'entier $K$ est un vecteur $\mathbf{p} \in \mathbb{N}^K$ tel que $\sum_{i=1}^{K} ip_i = K$.

Considérons $\mathcal{E} = \mathcal{G} = \mathbb{N}$, $\forall k \in \{1, 2, \ldots, M\},\ \mathcal{E}_k = \mathbb{N}$ et la suite $\mathcal{F}_{part} = (f_{part}^{(k)})_{k\in\mathbb{N}}$ définie telle que

$$f_{part}^{(k)}(\mathbf{p}) = \begin{cases} 0 & \text{si } k = 0 \\ \sum_{i=1}^{k} ip_i & \text{sinon.} \end{cases}$$

**Lemme 28.** *La suite $\mathcal{F}_{part} = (f_{part}^{(k)})_{k\in\mathbb{N}}$ vérifie la propriété sans mémoire.*

*Démonstration.* La fonction $f_{part}^{(0)}$ est constante et a pour valeur $0 \in \mathcal{G}$. Pour tout $k > 0$, la fonction $f_{part}^{(k)}$ a pour domaine de départ $\mathcal{E}_1 \times \ldots \times \mathcal{E}_k$ et pour ensemble d'arrivée $\mathcal{G}$. Pour tout $k > 0$, il existe bien une fonction $\oplus_k : \mathcal{G} \times \mathcal{E}_k \to \mathcal{G}$, elle est telle que

$$f_{part}^{(k)}(\mathbf{p}) = f^{(k-1)}(p_1, \ldots, p_{k-1}) \otimes p_k = f^{(k-1)}(p_1, \ldots, p_{k-1}) + kp_k.$$

□

Soit $K \in \mathbb{N}$ fixé, l'espace sans mémoire correspond à l'ensemble des partitions de $K$, en effet :

$$\Omega(K, K)_{part} = \left\{(p_1, p_2, \ldots, p_K) \in \mathbb{N}^K \mid p_1 + 2p_2 + \cdots + Kp_K = K\right\} := \mathcal{P}_E(K).$$

Comme $\mathcal{P}_E(K)$ correspond à l'espace sans mémoire $\Omega(K, K)_{part}$, on peut donc le représenter par un diagramme complet. Pour tout $\mathbf{p} \in \mathcal{P}_E(K)$ on a $f^{(0)}(\mathbf{p}) = 0$, ainsi le diagramme a pour nœud source $(0, 0)$. Il a pour nœud destination $(K, K)$. La fonction $g$ qui transforme une partition en un ensemble d'arcs est donnée par

$$\begin{array}{rccl} g & : & \mathcal{P}_E(M) & \to & \mathcal{P}(\mathcal{A}) \\ & & \mathbf{p} & \mapsto & \bigcup_{k=1}^{K} \left\{\left((k-1, \sum_{i=1}^{k-1} ip_i), (k, \sum_{i=1}^{k} ip_i)\right)\right\}. \end{array}$$

Considérons $D = (\mathcal{N}, A)$ un diagramme. Par définition la valeur d'un arc $a = \left((k-1, \ell), (k, \ell')\right) \in A$ est donnée par

$$v(a) := \left\{ p_k \in \mathbb{N} \mid \sum_{i=1}^{k-1} ip_i = \ell \text{ et } \sum_{i=1}^{k} ip_i = \ell' \right\} = \left\{ \frac{\ell' - \ell}{k} \right\}.$$

Pour tout arc $a \in A$, l'ensemble $v(a)$ est de cardinalité 1. On note ainsi $v(a) = n$ au lieu de $v(a) = \{n\}$. Le diagramme complet est donné par $\mathcal{D} := \mathcal{D}(K, K) = (\mathcal{N}, g(\mathcal{P}_E(K)))$.

**Exemple 36.** *La figure ci-dessous illustre le diagramme complet pour $K = 5$. En pointillé et rouge on lit la partition $\mathbf{x} = (1, 0, 0, 1, 0) \in \mathcal{P}_E(5)$.*

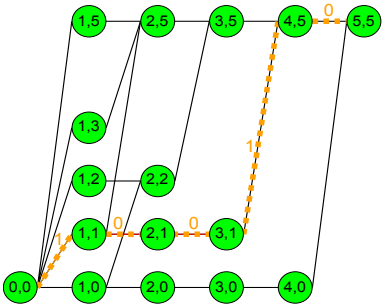



**FIGURE 7.4:** Diagramme complet, $K = 5$

**Lemme 29.** *Soit $D = (\mathcal{N}, A) \subseteq \mathcal{D}(K, K)$ un diagramme. Si $K \geq 2$, alors*

$$|A| \leq K + \sum_{k=2}^{K} \frac{K^2 - k^2 + K + k}{k}.$$

*Démonstration.* Soient $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ le diagramme complet et $D = (\mathcal{N}, A) \subseteq \mathcal{D}(K, K)$, alors $A \subseteq \mathcal{A}$. Majorons le nombre d'arcs du diagramme complet. La colonne $1$ possède $K$ arcs, en effet seul le nœud $(1, K-1)$ ne peut pas être atteint. En suivant ce raisonnement par récurrence, on en déduit que chaque colonne $k > 1$ du diagramme complet possède $K - k + 1$ nœuds de la forme $(k-1, \ell)$ et chaque nœud au plus $\frac{K}{k-\ell} + 1$ arcs sortants. Ainsi, $|\mathcal{A}(k)| \leq (K - k + 1)(\frac{K+k}{k}) = \frac{K^2 - k^2 + K + k}{k}$. □

La complexité de la représentation par diagramme est donc en $O(K^2 \log(K))$. Par exemple pour $K = 50$, on a $|\mathcal{P}_E(K)| = 204226$ et $|\mathcal{A}| = 3501$. Ce qui signifie qu'on peut encoder les $204226$ partitions de l'entier $50$ avec seulement $3501$ arcs.

### 7.2.2 Générateur de Boltzmann

L'algorithme 26 exposé à la fin de la section 7.1.3 est un générateur de Boltzmann pour $\textsc{mset}_K$. Il commence par choisir $\mathbf{p} \in \mathcal{P}_E(K)$ avec probabilité $\frac{A_{\mathbf{p}}(x)}{A_K(x)}$ puis génère le multi-ensemble selon la partition $\mathbf{p}$ choisie.

$$A_K(x) = \sum_{\mathbf{p} \in \mathcal{P}_E(K)} A_{\mathbf{p}}(x) \quad \text{avec} \quad A_{\mathbf{p}}(x) = \prod_{i=1}^{K} \frac{B(z^i)^{p_i}}{p_i! i^{p_i}} \text{ et } \mathbf{p} = (p_1, p_2, \ldots, p_k).$$

Remarquons que la distribution sur les partitions est à forme produit. Dans le chapitre 2 nous avons présenté l'algorithme `RandState` qui permet de produire un état $\mathbf{p} \in \psi(D)$ selon une distribution à forme produit donnée.

On propose ci-dessous un autre générateur de Boltzmann de paramètre $x$ pour les multi-ensembles de cardinalité fixe. Ce générateur est identique à celui proposé en section section 7.1.3 si ce n'est que la partition $\mathbf{p}$ est choisie à l'aide d'une marche aléatoire dans le diagramme complet $\mathcal{D}(K, K)$ et que le multi-ensemble est généré pendant la marche.

Soit $x$ dans le rayon de convergence de $B(z)$. Supposons que l'on dispose d'un oracle capable de fournir les valeurs $B(x^k)$ pour tout $k \in \mathbb{N}$. On peut alors en déduire $K$ fonctions telles que pour tout $k \in \{1, \dots, K\}$ :

$$\begin{array}{rccc} w_k & : & \mathbb{N} & \to & \mathbb{R}^+ \\ & & p & \mapsto & \frac{B(x^k)^p}{p!k^p}. \end{array}$$

On commence par attribuer pour chaque arc et chaque nœud du diagramme complet un poids à l'aide des algorithmes `WeightArcs` et `WeightNodes` du chapitre 2. Les fonctions poids calculées par `WeightArcs` et `WeightNodes` sont notées respectivement $w_A$ et $w_{\mathcal{N}}$. On a ainsi

$$w_A := \texttt{WeightArcs}(\mathcal{D}, w_1, \dots, w_K) \quad \text{et} \quad w_{\mathcal{N}} := \texttt{WeightArcs}(\mathcal{D}, w_1, \dots, w_K, w_A).$$

Notons que l'algorithme `WeightNodes` permet de calculer $A_K(x)$, en effet $w_{\mathcal{N}}((K, K)) = A_K(x)$.

L'algorithme 27 correspond à l'algorithme `RandState` de la section 2.2.6 dans lequel on a ajouté les lignes 7 et 8 pour la génération du multi-ensemble. Comme un arc $a = \big((k-1, \ell), (k, \ell')\big)$ ne représente ici qu'un seul $p_k$ et a pour valeur $v(a) = \frac{\ell' - \ell}{k}$, le chemin parcouru par l'algorithme 27 correspond à une unique partition.

**Algorithme 27:** $\Gamma_{\text{MSET}_K}[\mathcal{B}]$

**Données** : $D := \mathcal{D}(K, M)$, $w_A$, $w_{\mathcal{N}}$
**Résultat** : $\alpha \in \text{MSET}_K(\mathcal{B})$

1 **début**
  `/* Initialisation du multi-ensemble */`
2  $\alpha \leftarrow \emptyset$;
  `/* Marche aléatoire */`
3  $n_0 \leftarrow (0, 0)$;
4  **pour** $k = 1, 2, \dots, K$ **faire**
5  Choisir un arc $a = (n_{k-1}, n_k) \in A(k)$ avec probabilité $\frac{w_A(a) w_{\mathcal{N}}(n_k)}{w_{\mathcal{N}}(n_{k-1})}$;
6  $n \leftarrow v(a)$ ;
  `/* Construction du muti-ensemble */`
7  **pour** $c = 1, 2, \dots, n$ **faire**
8  $\alpha \leftarrow \alpha \uplus \Gamma_{\Delta_k}[\mathcal{B}](x)$;
9  **renvoyer** $\alpha$.

**Lemme 30.** *L'algorithme 27 est un générateur de Boltzmann pour* $\text{MSET}_M(\mathcal{B})$ *de paramètre* $x$.

*Démonstration.* Notons $\mathbf{p}$ la partition qui correspond a un chemin $u$ parcouru par l'algorithme 27. L'algorithme 27 correspond à l'algorithme `RandState`. On peut donc appliquer le lemme 8 et en déduire que la probabilité que le chemin $u$ soit parcouru est identique à la probabilité que `RandState` renvoie $\mathbf{p}$, c'est-à-dire $p_{\mathbf{p}}$ qui est telle que

$$\begin{cases} p_{\mathbf{p}} = \frac{1}{W} \prod_{i=1}^{K} w_k(p_i) = \frac{1}{A_K(x)} \prod_{i=1}^{K} \frac{B(x^i)^{p_i}}{p_i! i^{p_i}} = \frac{A_{\mathbf{p}}(x)}{A_K(x)} \\ W = \sum_{\mathbf{p} \in \mathcal{P}_E(K)} \prod_{i=1}^{K} w_k(p_i) = \sum_{\mathbf{p} \in \mathcal{P}_E(K)} A_{\mathbf{p}}(x) = A_K(x) \end{cases}.$$

L'algorithme 27 choisit donc un chemin qui correspond à une partition $\mathbf{p} \in \mathcal{P}_E(K)$ avec même probabilité que l'algorithme 26. De plus il génère le multi-ensemble qui est associé à $\mathbf{p}$ de la même façon que le ferait l'algorithme 26 qui aurait choisit $\mathbf{p}$. Comme l'algorithme 26

est un générateur de Boltzmann pour $\textsc{mset}_K(\mathcal{B})$, on en déduit que l'algorithme 27 est aussi un générateur de Boltzmann pour $\textsc{mset}_K(\mathcal{B})$.

□

## Conclusion

Dans ce chapitre on s'est intéressé à la génération de Boltzmann pour les multi-ensembles. On a notamment introduit le générateur de Boltzmann issu des travaux de [26] pour les multi-ensembles de cardinalité fixe. Il utilise les partitions d'entier pour déterminer le nombre et la taille des diagonales à générer pour produire un multi-ensembles.

On a proposé une autre application aux diagrammes : la représentation des partitions d'entiers. Nous avons ensuite utilisé le diagramme complet des partitions d'entier et une variante de l'algorithme `RandState` afin de proposer un autre générateur de Boltzmann pour les multi-ensemble de cardinalité fixe.

Ce chapitre clôt la partie théorique de cette thèse, dans le chapitre suivant on s'intéressera à l'implémentation des diagrammes. Notamment on présentera une bibliothèque Python qui implémente les partitions d'entiers ainsi que le générateur de Boltzmann pour les multi-ensembles de cardinalité fixe.

# Troisième partie

Implémentation

# 8 Logiciels

Chaque section de ce chapitre est consacrée à une production logicielle réalisée pendant cette thèse. Les sections sont présentées dans l'ordre chronologique de réalisation.

La première section présente la *toolbox* `Clones` (CLOsed queueing Networks Exact Sampling). Il s'agit d'une *toolbox* Matlab permettant la simulation du modèle présenté dans le chapitre 3 (réseaux fermés de files d'attente monoclasses, multiserveurs et à capacités finies). Dans `Clones` les diagrammes sont encodés par des matrices de taille $(M+1) \times K(M+1)$ et les diagrammes sans trou par des matrices de taille au plus $2 \times K(M+1)$. `Clones` a fait l'objet du papier [9] qui a reçu le prix *Best Tool Paper* à la conférence ValueTools 2014. Les sources ainsi qu'une documentation détaillée sont disponible à l'adresse `http://www.di.ens.fr/~rovetta/Clones`.

La seconde section est consacrée à `M-Clones` (Multiclass CLOsed queueing Networks Exact Sampling ) qui est un *package* Python permettant la simulation du modèle présenté dans le chapitre 4 (des réseaux fermés de files d'attentes muticlasses). Pour son implémentation on a choisi le paradigme orienté objet qui correspondait mieux au cas multiclasse. Le choix du langage Python a été effectué d'une part pour sa portabilité et d'autre part pour son ergonomie. Les sources sont disponibles à l'adresse `http://www.di.ens.fr/~rovetta/#Software`.

Enfin on présente un autre *package* Python `DiagramS`. Ce dernier a été conçu suite à la rédaction du chapitre 2 et utilise également le paradigme orienté objet. Son objectif est de proposer une implémentation générique des diagrammes. Un diagramme est vu comme un objet qui a pour attributs les paramètres $K$, $M$ ainsi qu'un autre objet représentant la suite sans mémoire associée au diagramme. Les algorithmes `CardStates`, `DiagramToStates` et `RandState` sont implémentés pour un diagramme quelconque. Ce *package* sera prochainement disponible sur le site `https://pypi.python.org/pypi` qui référence un grand nombre de *package* Python.

## 8.1 La toolbox `Clones`

`Clones` permet d'effectuer la simulation parfaite des réseaux fermés de files d'attente monoclasses à l'aide de diagrammes et de diagrammes sans trou. On commence par présenter les structures de données utilisées pour la représentation des diagrammes. On détaillera ensuite l'implémentation des fonctions de transition pour les deux types de diagramme. Enfin, on donnera des exemples pour l'utilisation de `Clones`.

### 8.1.1 Implémentation

Considérons $D = (\mathcal{N}, A) \subseteq \mathcal{D}(K, M)$ un diagramme au sens du chapitre 3. On distingue dans ce chapitre deux représentations : les diagrammes et les diagrammes sans trou. Regardons comment `Clones` encode $D$ dans ses deux cas. Pour la suite, lorsqu'il sera nécessaire de lever une ambiguïté, les diagrammes seront aussi appelés diagrammes "avec trou".

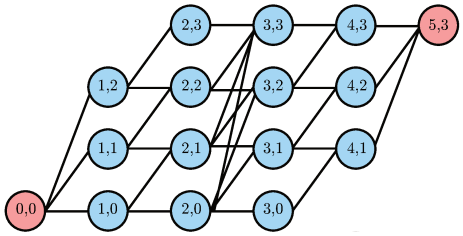



**FIGURE 8.1:** Diagramme complet avec $K = 5$, $M = 3$, et $\mathbf{C} = (2, 1, 3, 1, 2)$.

### Représentation matricielle

**Diagrammes "avec trou".** Soit $k \in \mathcal{Q}$ et $A(k)$ l'ensemble des arcs de la colonne $k$. Dans `Clones`, $A(k)$ est représenté par une matrice d'incidence notée $\mathtt{D_k}$, de taille $(M+1)\times(M+1)$. Pour $\ell, \ell' \in \mathbb{N}$, la présence de l'arc $\big((k-1,\ell),(k,\ell')\big)$ est encodée par $\mathtt{D_k}[\ell,\ell'] = 1$ et dans le cas contraire par $\mathtt{D_k}[\ell,\ell'] = 0$. Autrement dit, on a

$$\mathtt{D_k}[\ell,\ell'] = \mathbb{1}_{\left\{\big((k-1,\ell),(k,\ell')\big)\in A\right\}}.$$

On a vu au chapitre 3 qu'un arc $\big((k-1,\ell),(k,\ell')\big) \in A$ est tel que $\ell \leq \ell'$, ainsi $\mathtt{D_k}$ est une matrice triangulaire supérieure.

On appelle $v$-ième diagonale de la matrice $\mathtt{D_k}$ l'ensemble des coefficients $\mathtt{D_k}[\ell,\ell']$ tels que $\ell' - \ell = v$. Comme pour tout arc $a = \big((k-1,\ell),(k,\ell')\big) \in A(k)$, on a $bibera = \ell' - \ell$, la $v$-ième diagonale de la matrice $\mathtt{D_k}$ encode la présence des arcs de valeur $v$ en colonne $k$. Si $C_k$ est la capacité de la file $k$, alors pour tout arc $a \in A(k)$ on a $0 \leq bibera \leq C_k$ donc pour $v < 0$ et $v > C_k$, la $v$-ième diagonale est nulle.

Soient $i, j \in \mathcal{Q}$, on note $\mathtt{D_{i,j}}$ la matrice de taille $(M+1)\times(|i-j|+1)(M+1)$ qui correspond à la concaténation des matrices $\mathtt{D_k}$ pour $k$ compris entre $i$ et $j$, ainsi on a :

$$\mathtt{D_{i,j}} = \begin{cases} \mathtt{D_i}, \mathtt{D_{i+1}}, \ldots, \mathtt{D_j} & \text{si } i < j \\ \mathtt{D_j}, \ldots, \mathtt{D_{i-1}}, \mathtt{D_i} & \text{si } i > j \\ \mathtt{D_i} & \text{si } i = j. \end{cases}$$

Le diagramme $D$ est représenté par la matrice $\mathtt{D_{1,K}}$ de taille $(M+1)\times K(M+1)$ que l'on note simplement `D`. Dans la première colonne d'un diagramme, les arcs sont tous connectés au nœud source $(0,0)$, ainsi seule la première ligne de la matrice $\mathtt{D_1}$ n'est pas nulle. De la même manière, tous les arcs dans la dernière colonne d'un diagramme sont connectés au nœud destination, ainsi seule la dernière colonne de $\mathtt{D_K}$ n'est pas nulle.

**Exemple 37.** *La matrice ci-dessous encode le diagramme complet pour les paramètres $K = 5$, $M = 3$ et $\mathbf{C} = (2, 1, 3, 1, 2)$. Cette dernière est composée de 5 matrices booléennes de taille $4 \times 4$.*

```
D =
   1   1   1   0     1   1   0   0     1   1   1   1     0   1   0   0     0   0   0   0
   0   0   0   0     0   1   1   0     0   1   1   1     0   1   1   0     0   0   0   1
   0   0   0   0     0   0   1   1     0   0   1   1     0   0   1   1     0   0   0   1
   0   0   0   0     0   0   0   0     0   0   0   1     0   0   0   1     0   0   0   1
```

Le nombre d'arcs contenus dans une matrice D de taille $L \times C$ se calcule en additionnant les coefficients de la matrice, c'est-à-dire :

$$|A| = \sum_{\ell=1}^{L} \sum_{c=1}^{C} \mathtt{D}[\ell, c].$$

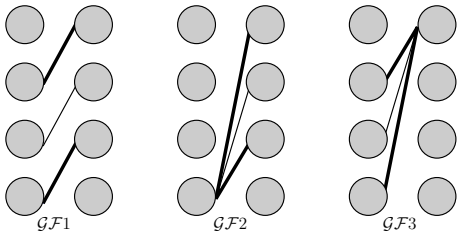



**FIGURE 8.2:** Conditions pour un diagramme sans trou.

**Diagrammes sans trou.** Un digramme sans trou est un diagramme satisfaisant les conditions $\mathcal{GF}1$, $\mathcal{GF}2$ et $\mathcal{GF}3$ (une illustration des conditions est rappelée en figure 8.2). Grâce à la condition $\mathcal{GF}1$, chaque colonne $k$ d'un diagramme sans trou peut être représentée par une matrice de taille $2 \times C_k + 1$ que l'on note $\mathtt{F_k}$. En effet, $\mathcal{GF}1$ implique que pour $v$ fixé, l'ensemble $L_v = \{\ell \mid \big((k-1,\ell),(k,\ell+v)\big) \in A(k)\}$ est un intervalle. Ainsi en encodant pour chaque valeur des arcs $v \in \{0, 1, \ldots, C_k\}$ le minimum et le maximum de l'intervalle $L_v$, on encode tous les arcs de la colonne $k$. On pose ainsi

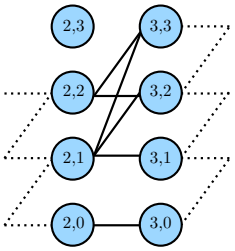
$$\begin{array}{rcl} \mathtt{F_k}[\ell, 0] & = & \min\big(\{\ell \mid \big((k-1,\ell),(k,\ell+v)\big) \in A(k)\}\big) \\ \mathtt{F_k}[\ell, 1] & = & \max\big(\{\ell \mid \big((k-1,\ell),(k,\ell+v)\big) \in A(k)\}\big), \end{array}$$

avec pour convention $\mathtt{F_k}[0, v] = \mathtt{F_k}[1, v] = -1$ si $\{\ell \mid \big((k-1,\ell),(k,\ell+v)\big) \in A(k)\} = \emptyset$.

Soient $i, j \in \mathcal{Q}$, comme dans le cas des diagrammes "avec trou", on note $\mathtt{F_{i,j}}$ la matrice de taille $2 \times (|i-j| + 1 + C_i + \ldots + C_j)$ qui correspond à la concaténation des matrices $\mathtt{F_k}$ pour $k$ compris entre $i$ et $j$. Un diagramme sans trou est représenté par la matrice $\mathtt{F_{1,K}}$ qui est de taille $2 \times (C_1 + C_2 + \ldots + C_K + K)$ ; on la note alors simplement F.

**Exemple 38.** *La matrice ci-dessous encode le diagramme complet pour les paramètres $K = 5$, $M = 3$ et $\mathbf{C} = (2, 1, 3, 1, 2)$. Il s'agit d'une matrice de taille $2 \times 14$. La figure 8.3 illustre la colonne d'un diagramme qui n'est pas le diagramme complet.*

```
F =
  0   0   0     0   0     0   0   0   0     1   0     3   2   1
  0   0   0     2   2     3   2   1   0     3   2     3   2   1
```




```
F3 =

      0     1     1    -1
      2     2     1    -1
```

**FIGURE 8.3:** Représentation d'une colonne d'un diagramme sans trou.

La valeur -1 représente l'absence d'arc. On peut compter le nombre d'arcs dans une matrice F de taille $2 \times C$ par la formule :

$$|A| = \sum_{v=1}^{C} (\mathtt{F}[1,v] - \mathtt{F}[0,v] + 1)\mathbb{1}_{\{\mathtt{F}[0,v] \geq 0\}}.$$

### Transition sur D.

Soit $D = (\mathcal{N}, A)$ un diagramme encodé par la matrice D, regardons comment implémenter la transition $T_{i,j,s}$ sur D. L'algorithme est donné dans la section 3.2 (algorithme 11).

Pour effectuer une transition sur un diagramme $D = (\mathcal{N}, A)$, dans l'algorithme 11, on commence par identifier trois sous ensembles d'arcs : $\mathcal{S}tay$, $\mathcal{F}ull$ et $\mathcal{T}ransit$. On calcule ensuite un nouvel ensemble d'arcs $\mathcal{T}ransit'$ à partir de $\mathcal{T}ransit$. Enfin, on retourne le diagramme $D' = (\mathcal{N}, \mathcal{S}tay \cup \mathcal{F}ull \cup \mathcal{T}ransit')$. La figure 8.4 illustre la transition $T_{4,2,1}$.

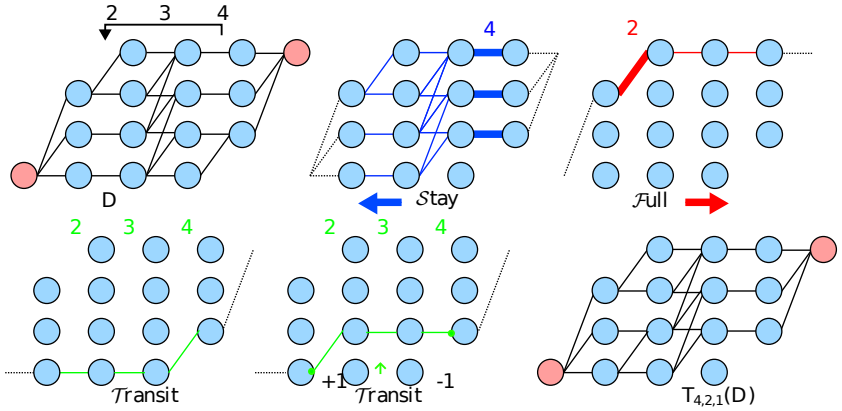



**FIGURE 8.4:** Transition $T_{4,2,1}$.

**Identifier les sous-ensembles d'arcs.** Soit $B \subseteq A(i)$ un-sous ensemble d'arcs en colonne $i$ contenant tous les arcs d'un ensemble de valeurs données $V \subseteq \{0, 1 \ldots, C_i\}$, c'est-à-dire :

$$B = \{a \in A(i) \text{ tel que } bibera \in V\}.$$

On note $\mathtt{D_i}^V$ la matrice de taille $(M+1) \times (M+1)$ qui représente $B$, ainsi on a :

$$\mathtt{D_i}^V[\ell, \ell'] = \mathtt{D_i}[\ell, \ell']\mathbb{1}_{\{\ell' - \ell \in V\}}.$$

Dans le chapitre 2, nous avons défini $\mathcal{P}ath(B)$ l'ensemble des arcs du diagramme pouvant être reliés à $B$ par

$$\mathcal{P}ath(B) = \bigcup_{a \in B} path(a) \quad \text{avec} \quad path(a) = \bigcup_{\mathbf{x} \in \psi(D) \mid a \in g(\mathbf{x})} g(\mathbf{x}).$$

On note $\mathtt{D_{i \to j}}^V$ la matrice booléenne de taille $(M+1) \times |i-j+1|(M+1)$ qui représente l'ensemble des arcs des colonnes $i, \ldots, j$ pouvant êtres reliés à $B$. La matrice $\mathtt{D_{i \to j}}^V$ représente ainsi l'ensemble $\mathcal{P}ath(B) \cap (A(i) \cup \ldots \cup A(j))$. Elle est composée de $|i-j|+1$ sous-matrices booléennes de taille $(M+1) \times (M+1)$ qui encodent les arcs des colonnes $i, \ldots, j$, ainsi on a :

$$\mathtt{D_{i \to j}}^V = [Y_1, Y_2, \ldots, Y_{|i-j|+1}].$$

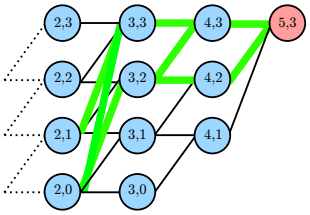
On note $\mathbf{1}_M$ la matrice de taille $(M+1)\times(M+1)$ dont toutes les composantes valent 1 et pour tout $u \in \{0,1\}^{M+1}$ on note $\texttt{diag}(u)$ la matrice dans laquelle on peut lire le vecteur $u$ sur sa diagonale et qui vaut 0 sinon. Ainsi pour tout $\ell, \ell' \in \{0,\dots,M\}$ on a : $\mathbf{1}_M[\ell,\ell'] = 1$ et $\texttt{diag}(u)[\ell,\ell'] = u_\ell \mathbb{1}_{\{\ell=\ell'\}}$.

La matrice $\texttt{D}_{\texttt{i}\to\texttt{j}}{}^V$ est construite récursivement de $i$ vers $j$ et selon le signe de $j-i$ de la façon suivante :

| Cas | $Y_1$ | $Y_k$ | $Y_{\lvert i-j\rvert+1}$ |
|---|---|---|---|
| $i<j$ | $\texttt{D}_\texttt{i}{}^V$ | $\texttt{diag}(\mathbf{1}_{M+1}Y_{k-1})\texttt{D}_\texttt{k}$ | |
| $i>j$ | $\texttt{D}_\texttt{k}\texttt{diag}(Y_{k+1}\mathbf{1}_{M+1}^t)$ | | $\texttt{D}_\texttt{i}{}^V$ |
| $i=j$ | $\texttt{D}_\texttt{i}{}^V$ | | |

Calculer $\texttt{diag}(\mathbf{1}_{M+1}Y_{k-1})\texttt{D}_\texttt{k}$ ne requiert pas de multiplication matricielle. Il suffit de copier les lignes de $\texttt{D}_\texttt{k}$ qui correspondent aux éléments non nul de la diagonale $\texttt{diag}(\mathbf{1}_{M+1}Y_{k-1})$. L'opération la plus complexe est de calculer $\mathbf{1}_{M+1}Y_{k-1}$, ce qui peut être fait en $O(M^2)$. Comme $|j-i|+1 \le K$, la complexité en temps de calcul pour trouver les chemins est en $O(KM^2)$.

La figure 8.5 illustre la matrice $\texttt{D}_{3\to5}{}^{\{2,3\}}$ et le diagramme correspondant. Les chemins (en gras et vert) correspondent aux chemins compris entre les colonnes 3 et 5 du diagramme pour lequel les valeurs des arcs en colonne 3 appartiennent à l'ensemble $\{2,3\}$.




$\texttt{D}_{3\to5}{}^{\{2,3\}} =$

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |

**FIGURE 8.5:** Matrice $\texttt{D}_{3\to5}{}^{\{2,3\}}$ correspondant aux arcs en gras et rouge.

**Calcul de $\mathcal{T}ransit'$.** Nous avons vu que l'ensemble $\mathcal{T}ransit'$ est obtenu à partir de $\mathcal{T}ransit$. À chaque arc $a_k := \big((k-1,\ell),(k,\ell')\big) \in \mathcal{T}ransit \cap A(k)$ correspond un arc $b \in \mathcal{T}ransit'$ qui est calculé de la façon suivante,

Si $i<j$, alors

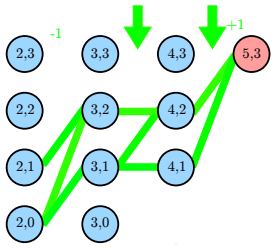
$$b_k = \begin{cases} a_k & \text{si } k<i \\ \big((i-1,\ell),(i,\ell'-1)\big) & \text{si } k=i \\ \big((k-1,\ell-1),(k,\ell'-1)\big) & \text{si } i<k<j \\ \big((j-1,\ell-1),(j,\ell')\big) & \text{si } k=j \\ a_k & \text{si } k>j \end{cases}$$

sinon ( *i.e.* $i>j$)

$$b_k = \begin{cases} a_k & \text{si } k<j \\ \big((j-1,\ell),(j,\ell'+1)\big) & \text{si } k=j \\ \big((k-1,\ell+1),(k,\ell'+1)\big) & \text{si } j<k<i \\ \big((i-1,\ell+1),(i,\ell')\big) & \text{si } k=i \\ a_k & \text{si } k>i. \end{cases}$$

Une transition revient à effectuer une translation de chaque matrice $\texttt{D}_\texttt{k}$ pour $k$ compris entre $i$ et $j$. Par exemple, imaginons que la matrice $\texttt{D}_{3\to5}{}^{\{2,3\}}$ de la figure 8.5 encode l'ensemble

$\mathcal{T}ransit$ pour la transition $T_{3,5,2}$ alors $\mathtt{D}_{\mathtt{T}'}$ qui encode $\mathcal{T}ransit'$ est donné en figure 8.6. Les 3 sous-matrices ont été déplacées respectivement suivant les translations $\leftarrow$, $\nwarrow$ et $\uparrow$.




$\mathtt{D}_{\mathtt{T}'}=$

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**FIGURE 8.6:** $\mathcal{T}ransit'$

Les opérations de translation des matrices sont décrites ci-dessous.

$$
\begin{array}{ll}
[\leftarrow X]_{\ell,\ell'} = X_{\ell,\ell'+1}\mathbb{1}_{\{\ell'<M\}}, & [\rightarrow X]_{\ell,\ell'} = X_{\ell,\ell'-1}\mathbb{1}_{\{\ell'>0\}}, \\
[\uparrow X]_{\ell,\ell'} = X_{\ell+1,\ell'}\mathbb{1}_{\{\ell<M\}}, & [\downarrow X]_{\ell,\ell'} = X_{\ell-1,\ell'}\mathbb{1}_{\{\ell>0\}}, \\
[\nwarrow X] = [\leftarrow[\uparrow X]], & [\searrow X] = [\rightarrow[\downarrow X]].
\end{array}
$$

**Algorithme.** Soient $X$ et $Y$ deux matrices booléennes on définit les opérations OU, ET sur les matrices à partir des opérateurs logiques ET ( AND en anglais) et OU ( OR en anglais).

$$
\begin{array}{rcl}
[X \text{ ET } Y][\ell,\ell'] & = & X[\ell,\ell'] \text{ ET } Y[\ell,\ell']. \\
[X \text{ OU } Y][\ell,\ell'] & = & X[\ell,\ell'] \text{ OU } Y[\ell,\ell'].
\end{array}
$$

Nous avons maintenant tous les outils nécessaires pour la traduction de l'algorithme 11 dans la représentation matricielle.

**Algorithme 28:** Transition $T(i,j,s,\mathtt{D})$

**Données :** $\mathtt{D}$, $(i,j) \in \mathcal{Q}$, $s \in \{1,\dots,E_k\}$

**début**
- $\mathtt{D}_{\mathtt{i},\mathtt{j}}{}^{\mathcal{S}tay} \leftarrow \mathtt{D}_{\mathtt{i}\rightarrow\mathtt{j}}{}^{\{0,1,\dots,s-1\}}$;
- $\mathtt{D}_{\mathtt{i},\mathtt{j}}{}^{\mathcal{F}ull} \leftarrow \mathtt{D}_{\mathtt{j}\rightarrow\mathtt{i}}{}^{\{C_j\}}$;
- $\mathtt{D}_{\mathtt{i},\mathtt{j}}{}^{\mathcal{T}ransit} \leftarrow \mathtt{D}_{\mathtt{i}\rightarrow\mathtt{j}}{}^{\{s,\dots,C_i\}}$ ET $\mathtt{D}_{\mathtt{j}\rightarrow\mathtt{i}}{}^{\{0,\dots,C_j-1\}}$;
- **si** $i < j$ **alors**
  - $\mathtt{D}'_{\mathtt{i}} \leftarrow [\leftarrow \mathtt{D}_{\mathtt{i}}^{\mathcal{T}\mathtt{ransit}}]$;
  - **pour** $k$ **de** $i+1$ **jusqu'à** $j-1$ **faire**
    - $\mathtt{D}'_{\mathtt{k}} \leftarrow [\nwarrow \mathtt{D}_{\mathtt{k}}^{\mathcal{T}\mathtt{ransit}}]$;
  - $\mathtt{D}'_{\mathtt{j}} \leftarrow [\uparrow \mathtt{D}_{\mathtt{j}}^{\mathcal{T}\mathtt{ransit}}]$;
  - $Y \leftarrow [\mathtt{D}'_{\mathtt{i}}, \mathtt{D}'_{\mathtt{i}+1}, \dots, \mathtt{D}'_{\mathtt{j}}]$;
- **sinon**
  - $\mathtt{D}'_{\mathtt{j}}{}^{\mathcal{T}\mathtt{ransit}} \leftarrow [\rightarrow \mathtt{D}_{\mathtt{j}}^{\mathcal{T}\mathtt{ransit}}]$;
  - **pour** $k$ **de** $j+1$ **jusqu'à** $i-1$ **faire**
    - $\mathtt{D}'_{\mathtt{k}}{}^{\mathcal{T}\mathtt{ransit}} \leftarrow [\searrow \mathtt{D}_{\mathtt{k}}^{\mathcal{T}\mathtt{ransit}}]$;
  - $\mathtt{D}'_{\mathtt{i}}{}^{\mathcal{T}\mathtt{ransit}} \leftarrow [\downarrow \mathtt{D}_{\mathtt{i}}^{\mathcal{T}\mathtt{ransit}}]$;
  - $Y \leftarrow [\mathtt{D}'_{\mathtt{j}}, \mathtt{D}'_{\mathtt{j}+1}, \dots, \mathtt{D}'_{\mathtt{i}}]$;
- $\mathtt{D}_{\mathtt{i},\mathtt{j}}{}' \leftarrow \mathtt{D}_{\mathtt{i},\mathtt{j}}{}^{\mathcal{S}tay}$ OU $\mathtt{D}_{\mathtt{j},\mathtt{i}}{}^{\mathcal{F}ull}$ OU $Y$;
- **renvoyer** $\mathtt{D}' = [\mathtt{D}_{1,\min(\mathtt{i},\mathtt{j})-1}, \mathtt{D}_{\mathtt{i},\mathtt{j}}{}', \mathtt{D}_{\max(\mathtt{i},\mathtt{j})+1,\mathtt{K}}]$.

Cet algorithme utilise $|i-j|+1$ matrices de taille $(M+1)\times(M+1)$. Chaque opération sur une matrice se fait avec au plus $O(M^2)$ opérations. Comme $|i-j|+1 \leq K$, l'algorithme 28 requiert $O(KM^2)$ opérations.

**Transition sur $\mathtt{F}$.**

L'algorithme de transition sur une matrice $\mathtt{F}$ représentant un diagramme sans trou suit les mêmes étapes que l'algorithme 28 pour une matrice "avec trou" $\mathtt{D}$. Cependant nous avons besoin de redéfinir les opérations sur les matrices représentant les colonnes, à savoir l'identification des chemins, les opérations (ET, OU) et les translations et ($\leftarrow$, $\rightarrow$, $\uparrow$, $\downarrow$, $\nwarrow$, $\searrow$). On considère maintenant uniquement des matrices de taille $2 \times (1 + C)$.

**Opérations booléennes.** Les opérations ET et OU correspondent ici respectivement à une intersection et à une union d'intervalle. Pour tout $\ell \in \{1, 2\},\ v \in \{0, 1, \dots, C\}$, on pose

$$[\mathtt{X}\ \text{ET}\ \mathtt{Y}][\ell, v] = \begin{cases} -1 & \text{si } \mathtt{X}[1, v] = -1 \text{ ou } \mathtt{Y}[1, v] = -1 \\ & \text{ou } \max(\mathtt{X}[1, v], \mathtt{Y}[1, v]) < \min(\mathtt{X}[2, v], \mathtt{Y}[2, v]) \\ \max(\mathtt{X}(1, v), \mathtt{Y}(1, v)) & \text{sinon et si } \ell = 1 \\ \min(\mathtt{X}(2, v), \mathtt{Y}(2, v)) & \text{sinon et si } \ell = 2, \end{cases}$$

et

$$[\mathtt{X}\ \text{OU}\ \mathtt{Y}][\ell, v] = \begin{cases} \mathtt{X}[\ell, v] & \text{si } \mathtt{Y}[1, v] = -1 \\ \mathtt{Y}[\ell, v] & \text{si } \mathtt{X}[1, v] = -1 \\ \min(\mathtt{X}[1, v], \mathtt{Y}[1, v]) & \text{sinon et si } \ell = 1 \\ \max(\mathtt{X}[2, v], \mathtt{Y}[2, v]) & \text{sinon et si } \ell = 2. \end{cases}$$

**Translations des matrices.** On redéfinit maintenant les translations $[\leftarrow \mathtt{X}]$,$[\rightarrow \mathtt{X}]$, $[\uparrow \mathtt{X}]$, $[\downarrow \mathtt{X}]$, $[\nwarrow \mathtt{X}]$, $[\searrow \mathtt{X}]$ par

$$[\leftarrow \mathtt{A}][\ell, v] = \begin{cases} -1 & \text{si } v = C \\ \mathtt{A}[\ell, v + 1] & \text{sinon.} \end{cases} \qquad [\rightarrow \mathtt{A}][\ell, v] = \begin{cases} -1 & \text{si } v = 0 \\ \mathtt{A}[\ell, v - 1] & \text{sinon.} \end{cases}$$

$$[\uparrow \mathtt{A}][\ell, v] = \begin{cases} -1 & \text{si } v = 0 \\ -1 & \text{si } \mathtt{A}[\ell, v - 1] = -1 \\ 0 & \text{si } \mathtt{A}[\ell, v - 1] = 0 \\ \mathtt{A}[\ell, v - 1] - 1 & \text{sinon.} \end{cases} \qquad [\downarrow \mathtt{A}][\ell, v] = \begin{cases} -1 & \text{si } v = C \\ -1 & \text{si } \mathtt{A}[\ell, v + 1] = -1 \\ \mathtt{A}[\ell, v + 1] + 1 & \text{sinon.} \end{cases}$$

$$[\nwarrow \mathtt{A}][\ell, v] = \begin{cases} -1 & \text{si } \mathtt{A}[1, v] = -1 \\ -1 & \text{si } \mathtt{A}[2, v] = 0 \\ \max(\mathtt{A}[\ell, v] - 1, 0) & \text{sinon.} \end{cases} \qquad [\searrow \mathtt{A}][\ell, v] = \begin{cases} -1 & \text{si } \mathtt{A}[1, v] = -1 \\ \min(\mathtt{A}[\ell, v] + 1, C - v) & \text{sinon.} \end{cases}$$

**Identifier les chemins.** On cherche à définir la matrice ${\mathtt{F}_{\mathtt{i}\to\mathtt{j}}}^V$, l'équivalent à la matrice ${\mathtt{D}_{\mathtt{i}\to\mathtt{j}}}^V$.

Pour $V \subseteq \{0, 1, \dots, C_k\}$ un ensemble de valeurs, on pose ${\mathtt{F}_\mathtt{k}}^V$ la matrice qui représente la présence des arcs à valeur dans $V$ dans la colonne $k$ par :

$${\mathtt{F}_\mathtt{k}}^V[\ell, v] = \mathtt{F}_\mathtt{k}[\ell, v]\mathbb{1}_{\{v \in V\}} - \mathbb{1}_{\{v \notin V\}}.$$

L'algorithme 29 trouve les chemins dans un diagramme pour la représentation matricielle sans trou.

Le $\ell$ minimal à considérer pour les nœuds $(k, \ell)$ dans la colonne suivante est noté $v_1$ et le $\ell$ maximal $v_2$. La figure 8.7 illustre la première étape de l'algorithme pour $i < j$.

**Algorithme 29:** Trouver les chemins pour la représentation sans trou

**Données** : $\mathtt{F_i}^V$
**Résultat** : $\mathtt{F_{i\to j}}^V$

**début**
  **si** $i < j$ **alors**
    $c \leftarrow 1$;
    $Y_c \leftarrow \mathtt{F_i}^V$;
    **pour** $k$ **de** $i+1$ **jusqu'à** $j$ **faire**
      $W \leftarrow \{v | Y_c(1,v) > -1\}$ ;
      $v_1 \leftarrow \min(\{Y_c(1,v)+v) | v \in W\})$;
      $v_2 \leftarrow \max(\{Y_c(2,v)+v) | v \in W\})$;
      $c \leftarrow c+1$ ;
      **pour** $v$ **dans** $W$ **faire**
        $Y_c(1,v) \leftarrow \max(v_1, \mathtt{F_k}(1,v))$;
        $Y_c(2,v) \leftarrow \min(v_2, \mathtt{F_k}(2,v))$;
  **si** $i > j$ **alors**
    $c \leftarrow |i-j|+1$;
    $Y_c \leftarrow \mathtt{F_i}^V$;
    **pour** $k$ **de** $i+1$ **jusqu'à** $j$ **faire**
      $W \leftarrow \{v | Y_c(1,v) > -1\}$ ;
      $v_1 \leftarrow \min(\{Y_c(1,v) | v \in W\})$;
      $v_2 \leftarrow \max(\{Y_c(2,v) | v \in W\})$;
      $c \leftarrow c-1$;
    **pour** $v$ **dans** $W$ **faire**
      $Y_c(1,v) \leftarrow \max(v_1+v, \mathtt{F_k}(1,v))$;
      $Y_c(2,v) \leftarrow \min(v_2+v, \mathtt{F_k}(2,v))$;
  $\mathtt{F_{i\to j}}^V \leftarrow [Y_1, Y_2, \ldots, Y_{|i-j|+1}]$;
  **renvoyer** $\mathtt{F_{i\to j}}^V$.

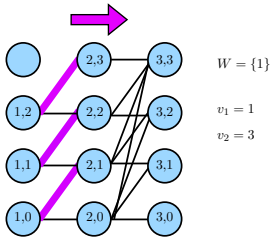



**FIGURE 8.7:** Chemins commençant par des arcs de valeur 1 dans la colonne 2.

**Algorithme.** L'algorithme de transition opérant sur $\mathtt{F_k}$ est le même que celui opérant sur $\mathtt{D_k}$ (l'algorithme 28). Il suffit de remplacer les opérations sur les matrices que nous venons de définir : l'identification des chemins, les opérations (ET, OU) et les translations et ($\leftarrow$,$\rightarrow$, $\uparrow$, $\downarrow$, $\nwarrow$, $\searrow$). La transition est effectuée en considérant $|i-j|+1$ matrices, chaque opération sur chaque matrice peut être effectuée avec une complexité en temps de calcul en $O(M)$. Comme $|i-j|+1 \leq K$, l'algorithme requiert au plus $O(KM)$ opérations élémentaires.

## 8.1.2 Utilisation de `Clones`

Dans `Clones` on peut utiliser au choix : des diagrammes avec ou sans trou ou bien des ensembles d'états. On peut choisir de les représenter directement par des matrices ou par des objets. Nous décrivons ici comment utiliser `Clones` par le paradigme orienté objet. Une documentation plus approfondie est disponible à l'adresse suivante `http://www.di.ens.fr/~rovetta/Clones`. Pour illustrer cette présentation, on considère le réseau donné en figure 8.8 contenant $M = 3$ clients, ayant pour vecteur des capacités $\mathbf{C} = (2, 1, 3, 1, 2)$ et pour vecteur du nombre de serveurs $\mathbf{E} = (2, 1, 2, 1, 2)$.

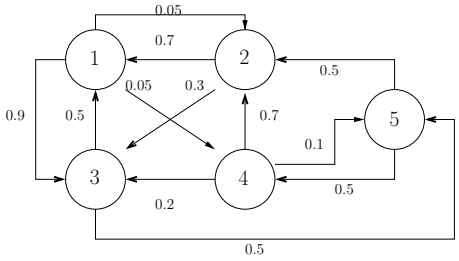



**FIGURE 8.8:** Réseau fermé avec 5 files.

Avant de commencer à construire les diagrammes et faire les simulations, on doit définir le modèle, c'est à dire le nombre de files, de clients, de serveurs dans chaque file, le taux de service et la matrice de routage. C'est ce que propose la classe `Clones_Model`.

### La classe `Clones_Model`.

Le tableau 8.1 illustre tous les paramètres de notre modèle. Puisque `Clones` est développé en `Matlab`, ces derniers sont des matrices.

La classe `Clones_Model` contient tous les paramètres. Le réseau de l'exemple 8.8 est décrit par l'objet `ModEx1` en utilisant la commande :

```
>> ModEx1=Clones_Model('Example 1',C,S,Mu,P,M);
```

| Nom | Taille | Type | Représentant | Valeur |
|---|---|---|---|---|
| K | $1 \times 1$ | int | Nombre de file | 5 |
| M | $1 \times 1$ | int | Nombre de clients | 4 |
| C | $1 \times K$ | int | Capacité dans chaque file | [2 1 3 1 2] |
| S | $1 \times K$ | int | Nombre de serveurs dans chaque file | [2 1 2 1 2] |
| Mu | $1 \times K$ | double | Taux de service pour chaque serveur dans chaque file | [1 1 5 1 1] |
| P | $K \times K$ | double | Matrice de routage | $\begin{bmatrix} 0 & 0.05 & 0.9 & 0.05 & 0 \\ 0.7 & 0 & 0.3 & 0 & 0 \\ 0.5 & 0 & 0 & 0 & 0.5 \\ 0 & 0.7 & 0.2 & 0 & 0.1 \\ 0 & 0.5 & 0 & 0.5 & 0 \end{bmatrix}$ |

**TABLE 8.1:** Paramètres du modèle choisi.

### Les classes `Clones_D` et `Clones_F`.

La classe `Clones_D` représente un diagramme, elle a pour attribut (entre autres) une matrice de type `D`. La figure 8.9 illustre la création de l'objet `Diagram` correspondant au modèle `ModEx1`. Elle est crée en utilisant le constructeur `Clones_D`. La méthode `Tm` effectue la transition directement sur le diagramme, on calcule $D = T_{4,2,1} \circ T_{4,5,1} \circ T_{4,5,1} \circ T_{2,1,1}(\mathcal{D})$ (le diagramme de la figure 8.4). On dessine ensuite le diagramme ainsi obtenu à l'aide de la méthode `plot` et on affiche les états de l'ensemble $\psi(D)$ grâce à la méthode `Psi`.

```
>> Diagram=Clones_D(ModEx1);
>> Diagram=Tm(Diagram,[2 1 1]);
>> Diagram=Tm(Diagram,[4 5 1]);
>> Diagram=Tm(Diagram,[4 5 1]);
>> Diagram=Tm(Diagram,[4 2 1]);
>> plot(Diagram,-1);
>> SD=Psi(Diagram)
SD =
     0     0     1     0     2
     0     1     0     0     2
     1     0     0     0     2
     0     0     2     0     1
     0     1     1     0     1
     1     0     1     0     1
     2     0     0     0     1
     0     0     3     0     0
     0     1     2     0     0
     1     0     2     0     0
     2     0     1     0     0
     2     1     0     0     0
```

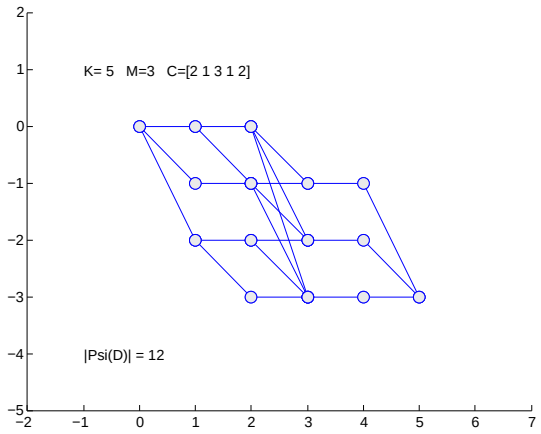



**FIGURE 8.9:** Exemple d'utilisation de la classe `Clones_D`.

`Clones_F` est l'analogue à la classe `Clones_D` pour la représentation sans trou. Dans la figure 8.10, on commence par construire l'objet `GapFree` qui implémente le diagramme complet $\mathcal{D}$ en accord avec les paramètres contenus dans `ModEx1`. On calcule ensuite $D = T^{\mathcal{GF}}_{3,5,2}(\mathcal{D})$ et on le dessine. Pour finir, on construit `Diagram` de type `Clones_D` qui correspond à une représentation "avec trou" de l'objet `GapFree`. Ceci ce fait en utilisant la méthode `FtoD`.

### Simulation.

La fonction `Clones_rand_ijs` renvoie le triplet aléatoire $r = [i, j, s]$ pour lequel $i$ est la file de départ, $j$ celle d'arrivée et $s$ le nombre minimal de clients qu'il doit y avoir en file $i$ pour

```
>> GapFree=Clones_F(ModEx1);
>> GapFree=gfTm(GapFree,[3 5 2]);
>> plot(GapFree,-1);
>> Diagram=FtoD(GapFree);
```

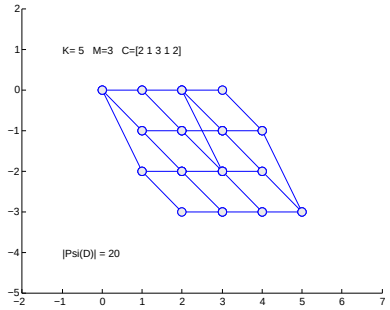



**FIGURE 8.10:** Exemple d'utilisation de la classe `Clones_F`.

qu'un client soit servi. La distribution de $r$ dépend des paramètres de la fonctions qui sont : `P`, `E` et `Mu`. La fonction `Clones_rand_ijs` est utilisée dans les méthodes `PSS`, `PSD` et `PSF` de la classe `Clones_Model`. Ces méthodes produisent un échantillon de la distribution stationnaire du modèle et utilisent respectivement les algorithmes de simulation parfaite du même nom dans le chapitre 3. Par exemple, pour produire $2$ échantillons suivant l'algorithme 12 correspondant au modèle décrit par `ModEx1`, on effectue la commande suivante :

```
 >> PSD(ModEx1,2)
ans =
     0     0     0     1     2
     2     0     0     0     1
```

## 8.2 Le package `M-Clones`

Le package `M-Clones` contient $3$ modules : `model` (pour la définition du modèle), `state` (pour manipuler les ensembles d'états comme des vecteurs) et `diagram`. Dans ce dernier figurent les classes `Diagram`, `Column` et `Arc` qui encodent respectivement un diagramme, un ensemble d'arcs pour un $k$ donné et un arc. Plusieurs méthodes sont définies dans la classe `Diagram`, en voici les principales : `plot` dessine le diagramme pour $Z \leq 3$, `psi` retourne l'ensemble d'états contenus dans diagramme, `card_psi` retourne le nombre d'états contenus dans diagramme, `T` réalise une transition sur le diagramme, `reset` transforme le diagramme courant en un diagramme contenant tous les états et `exactSample` utilise l'algorithme `PSM` du chapitre 4 pour produire un état distribué selon la distribution stationnaire.

Les colonnes du diagramme sont encodées par un tableau de type `list` et de longueur $K$. Chaque case de ce tableau contient un objet de type `Column`. Les classes `Column` et `Arc` sont des métaclasses et héritent donc d'un `type` Python. Elles bénéficient ainsi des méthodes (nombreuses et optimisées) définies pour leur type. La classe `Column` hérite du type `set` et contient un ensemble d'objets de type `Arc` eux-mêmes définis à partir du type `tuple`.

`M – Clones` est disponible à l'adresse `http://www.di.ens.fr/~rovetta/#Software`.

## 8.3 Le package `DiagramS`

Le *package* `DiagramS` met en œuvre les objets et les algorithmes du chapitre 2. À l'instar de `M – Clones`, `DiagramS` est implémenté en Python. Il utilise la programmation par objet et la programmation fonctionnelle.

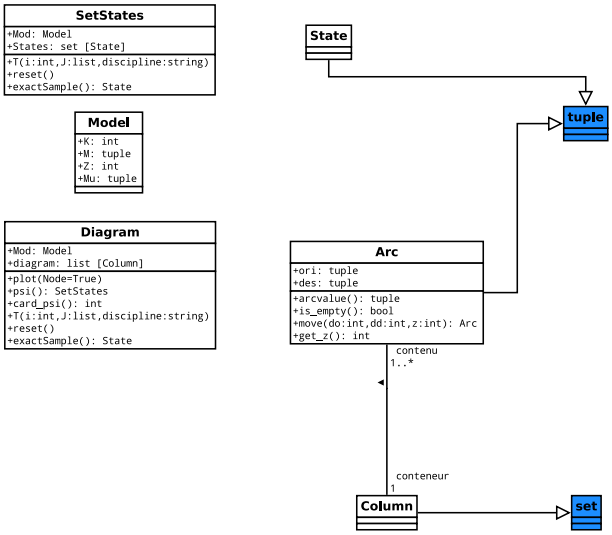



**FIGURE 8.11:** Représentation UML `M-Clones`.

La figure 8.12 représente le diagramme UML (non exhaustif) des principales classes et méthodes de ce *package*. Les deux objets centraux sont `MLS` et `Diagram`, ils implémentent respectivement les suites sans mémoire et les diagrammes.

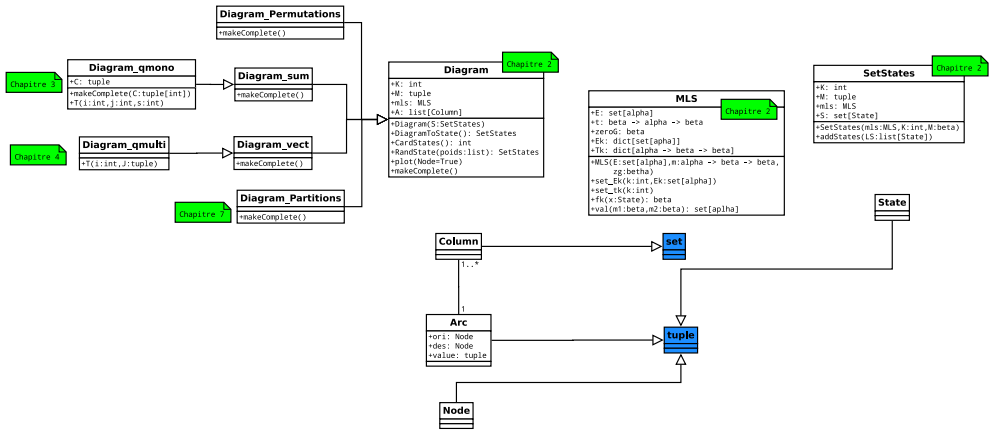



**FIGURE 8.12:** Représentation UML des classes du package `DiagramS`.

## 8.3.1 Le module `memorylesseq`

La classe `MLS` implémente une suite sans mémoire en se basant sur les résultats de la section 2.1. Pour représenter $\mathcal{F} := (f^{(k)})_{k\in\mathbb{N}}$ une suite sans mémoire, il faut spécifier : l'espace $\mathcal{E}$, la suite $(\mathcal{E}_k)_{k>0}$, la suite $(\oplus_k)_{k\in\mathbb{N}}$ et $0_{\mathcal{G}}$. Il n'est pas nécessaire de spécifier $\mathcal{G}$.

Dans `DiagramS`, l'espace $\mathcal{E}$ est représenté par un ensemble d'objets de type `alpha`. Si pour $k > 0$, le dictionnaire `DEk` possède la clef $k$ alors $\mathcal{E}_k$ a pour valeur `DEk[k]` sinon il a pour valeur `E`. De même, si le dictionnaire `Dotk` possède la clef $k$ alors $\oplus_k$ a pour valeur `Dotk[k]` sinon elle a pour valeur la fonction `ot`. La programmation fonctionnelle est utilisée afin de définir les fonctions $\oplus_k$.

| Notation Chap. 2 | Nom et domaine dans MLS | Commentaires |
|---|---|---|
| $\mathcal{E}$ | `E: set[alpha]` | Valeur par défaut pour $\mathcal{E}_k$ |
| $\oplus : \mathcal{G} \to \mathcal{E} \to \mathcal{G}$ | `ot: beta -> alpha -> beta` | Fonction par défaut pour $\oplus$ |
| $0_{\mathcal{G}}$ | `zG: beta` | Valeur de type `beta` |
| $(\mathcal{E}_k)_{k>0}$ | `DEk: [int,alpha]` | Dictionnaire pour $\mathcal{E}_k$ |
| $(\oplus_k)_{k>0}$ | `Dotk: [int,beta->alpha->beta]` | Dictionnaire pour les fonctions $\oplus_k$ |

**TABLE 8.2:** Attributs de la classes `MLS`.

| Notation Chap. 2 | Signatures | Commentaires |
|---|---|---|
| | `MLS(E,ot,zg)` | Constructeur |
| | `set_Ek(k:int, Ek)` | Affecte à $\mathcal{E}_k$ l'ensemble `Ek` |
| | `set_Otk(k:int,otk)` | Affecte à $\oplus_k$ la fonction `otk` |
| $f^{(k)}$ | `fk(x:State,k:int)` | Renvoie la valeur $f^{(k)}(\mathbf{x})$ |
| $v$ | `val(k:int,m1:beta,m2:beta)` | Renvoie la valeur de l'arc $\big((k-1,m1),(k,m2)\big)$. |

**TABLE 8.3:** Méthodes de la classes `MLS`.

### 8.3.2 Le module `diagram`

La classe `Diagram` implémente une suite sans mémoire en se basant sur les résultats de la section 2.2. Un diagramme a pour attributs : une suite sans mémoire, un nombre de colonne $K$, la valeur du nœud destination $M$ et une liste de colonnes qui contiennent des arcs. On utilise la même structure de données que dans `M-Clones` : un diagramme est représenté par une liste de colonnes (classe `Column`), une colonne est un ensemble d'arcs (classe `Arc`), un arc est un couple de nœuds (classe `Node`).

Les méthodes sont détaillées dans le tableau 8.4, elles implémentent les algorithmes du chapitre 2.

| Algo. Chap. 2 | Signature | Commentaires |
|---|---|---|
| | `Diagram(K:int,E:set[alpha],mls:MLS)` | Constructeur |
| `CardStates` | `CardStates():int` | Renvoie $\lvert\psi(D)\rvert$ |
| `DiagramToStates` | `DiagramToStates():SetStates` | Renvoie $\psi(D)$ |
| `StatesToDiagram` | `StatesToDiagram(S:SetStates)` | Effectue $D = \phi(S)$ |
| `RandState` | `RandState(w:dict[tuple[int,alpha],float]):State` | Génère $\mathbf{x} \in \psi(D)$ |
| | `plot()` | Dessine le diagramme |
| | `makeComplete()` | Rend le diagramme complet |

**TABLE 8.4:** Méthodes de la classes `MLS`.

### 8.3.3 Exemples d'utilisations

Montrons comment implémenter un diagramme complet des partitions de $50$. On commence par construire la suite sans mémoire `mls`. Pour $k > 0$, on effectue les affectations suivantes : $\mathcal{E}_k = \{0, k, 2k, \dots, \}$ et $\oplus_k := x \times y \mapsto x + ky$. Une fois la suite construite on construit le diagramme des partitions de $50$.

```
# Ensemble E
E=set(range(51))
```

```
# Suite sans memoire
ot = lambda x, y : 0
mls=MLS(E,ot,0)
for k in xrange(1,K+1):
~~~mls.setEk(range(0,51,k))
~~~mls.setOtk(lambda x,y : x+k*y)

# Diagramme complet des partitions
D=Diagram(50,50,mls)
D.plot()
```

Bien que l'on puisse construire un diagramme des partitions en définissant une suite sans mémoire, la classe `Diagram_Partition` permet de construire directement des diagrammes complets de partition. Elle hérite de la classe `Diagram` et redéfinit certaines fonctions afin d'optimiser les performances. Le code ci-dessous produit le même diagramme que le code ci-dessus.

```
D=Diagram_Partition(50)
D.plot()
```

La figure 8.13 illustre deux diagrammes complets obtenus à l'aide du package `DiagramS` en utilisant la méthode `plot`. Dans celui de gauche on peut lire les $204226$ partions de l'entier $50$, c'est le diagramme $\mathcal{D} = (\mathcal{N}, \mathcal{P}_E(50))$ du chapitre 7 que l'on a construit avec `Diagram_Partition`. Le diagramme de droite est le diagramme complet de toutes les permutations de longueur $7$ et contient donc $7!$ chemins (voir chapitre 2) on le construit avec `Diagram_Permutation`.

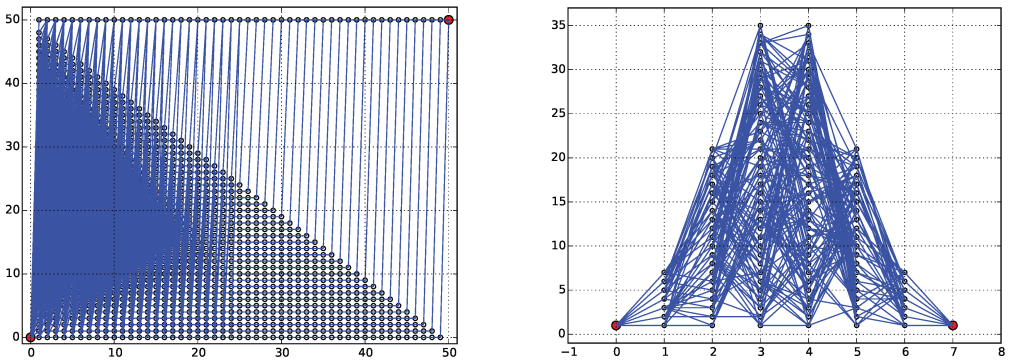



**FIGURE 8.13:** Exemples de diagrammes complets obtenus avec `DiagramS`

Le *package* `DiagramS` permet de calculer le nombre d'états contenus dans un diagramme par la méthode `CardStates`. Il permet de plus la génération aléatoire d'états avec la méthode `RandStates`.

# Conclusion

La *toolbox* `Clones` fut la première à être développée, la représentation matricielle est apparue comme la plus simple et la plus rapidement implémentable. En effet dans `Clones`, les transitions s'effectuent en translatant des matrices, en les additionnant modulo 2 ou par des simples recherches du $\min$ et du $\max$ dans une matrice. Cependant cette représentation matricielle n'a pas pu être adaptée de manière aussi efficace pour `M` − `Clones`. On a alors choisi le paradigme orienté objet pour la structure du diagramme. Cette nouvelle représentation

s'adapte très bien à tout type de diagramme c'est pourquoi cette même idée est appliquée pour l'implémentation de la structure des diagrammes dans `DiagramS`.

Les outils `Clones` et `M − Clones` sont dédiés à la simulation parfaite des réseaux fermés de files d'attentes, à la différence de `DiagramS` qui a pour objectif la construction des diagrammes. Ce dernier se base sur la théorie développée au chapitre 2 et pourra être enrichi en fonction des avancées théorique concernant les diagrammes. Par exemple, l'implémentation des diagrammes valués dans `DiagramS` permettrait la simulation des chaînes de Markov à horizon fixé.

# Conclusion et perspectives

La contribution majeure de cette thèse est la mise en œuvre d'une nouvelle technique de simulation parfaite pour les réseaux fermés de files d'attente. Elle repose sur la représentation par un graphe de l'espace des états, appelé *diagramme*, ainsi que sur un algorithme de transition qui permet la mise à jour simultanée de tous les états représentés par un tel diagramme. Cette technique a été appliquée dans un premier temps à des réseaux monoclasses à capacité finie et a permis de réaliser des transitions avec une complexité polynomiale en le nombre de files et de clients.

L'utilisation de diagrammes pour la simulation parfaite a été généralisée pour des modèles de réseaux multiclasses. On a construit l'algorithme de transition et démontré l'existence d'une suite couplante de transitions. On s'est ensuite intéressé à la simulation parfaite pour deux modèles de réseaux possédant des synchronisations. Pour ce faire, on a défini les *diagrammes agglomérés* qui sont composés de plusieurs bouts de diagrammes représentant chacun un sous-réseau du modèle étudié. On a exhibé pour ces deux modèles un algorithme de transition ainsi qu'une suite couplante de transitions, ce qui a permis une nouvelle fois la mise en œuvre de notre technique de simulation parfaite.

À l'origine, la structure de données *diagramme* fut conçue afin de pouvoir s'adapter aussi bien aux réseaux monoclasses que multiclasses. Dans le cadre des diagrammes, on a introduit les *espaces sans mémoire* qui caractérisent les ensembles d'objets ayant les propriétés qui permettent d'être représentés par un diagramme. Cette définition englobe par ailleurs des objets combinatoires autres que les espaces d'états des réseaux fermés de files d'attente, notamment les permutations et les partitions d'entiers. Des algorithmes utilisant le paradigme de programmation dynamique et agissant sur les diagrammes ont été conçus pour compter le nombre d'états représentés par un diagramme, pour transformer un diagramme en un espace d'états et pour générer aléatoirement les états encodés dans un diagramme selon une distribution à forme produit.

La deuxième partie de cette thèse fut consacrée à la combinatoire analytique et aux générateurs de Boltzmann. L'espace des états d'un réseau fermé de files d'attente monoclasse a été étudié comme une classe combinatoire et on a proposé un générateur de Boltzmann pouvant échantillonner la distribution stationnaire d'un réseau fermé monoclasse à forme produit pour un nombre de clients non fixé. Pour consolider ce résultat, une étude concernant la distribution du nombre de clients renvoyé par le générateur pourrait être menée ; de plus, l'échantillonneur pourrait être étendu pour les réseaux fermés multiclasses à forme produit.

Une nouvelle utilisation des diagrammes a été proposée via la construction d'un générateur de Boltzmann pour les multi-ensembles de cardinalité fixe. Ce générateur utilise le diagramme complet des partitions de l'entier qui fixe la cardinalité du multi-ensemble et effectue une marche aléatoire dans le diagramme.

La *toolbox* Matlab `Clones` a été implémentée afin de mettre en œuvre la technique de simulation parfaite du chapitre 3. Dans `Clones`, les diagrammes sont encodés par des matrices d'incidence. Cette *toolbox* a également fait l'objet de l'article [9], lequel a reçu le prix *Best Tool Paper Award* à la conférence ValueTools 2014.

Le *package* Python `DiagramS` implémente les diagrammes et les algorithmes en adoptant les principes du chapitre 2 et en utilisant la programmation orientée objets. Pour construire un objet de type diagramme, l'utilisateur doit simplement définir la suite sans mémoire et préciser les paramètres souhaités : le *package* construit alors le diagramme correspondant. Les algorithmes du chapitre 2, comme la génération aléatoire, sont implémentés par des méthodes agissant sur les diagrammes.

La technique qui permet d'effectuer la simulation parfaite devrait pouvoir être exploitée prochainement sur d'autres types de réseaux fermés de files d'attente, notamment comme le laissent penser les résultats du chapitre 5, à d'autres types de réseaux possédant une ou plusieur(s) synchronisation(s). Elle devrait de plus être facilement exploitable pour des réseaux fermés dans lesquels les clients arrivent par lots. Les modifications des pentes des arcs du diagramme correspondraient alors à la taille du lot considéré. Par la suite, on pourrait s'intéresser à d'autres problèmes, comme les réseaux fermés possédant des clients négatifs [29, 31] et les tables de contingences [32] dont les cellules rappellent les états des réseaux fermés multiclasses.

Le chapitre 3 abordait la problématique du temps de couplage des diagrammes d'un point de vue expérimental : on a proposé deux stratégies visant à rapprocher le temps de couplage du diagramme à celui de la chaîne qui modélise le réseau étudié. La première stratégie visait à trouver une bonne numérotation du réseau ; la fonction qui évalue cette bonne numérotation est une heuristique simple et pourrait sans doute être améliorée. On pourrait par exemple envisager des heuristiques se basant sur les algorithmes de recherche des arbres couvrants maximaux et de recherche de chemins de poids maximal. La seconde stratégie consistait à commencer la simulation avec les diagrammes puis à la poursuivre avec un ensemble d'états. Là encore, cette piste mérite d'être approfondie, notamment pour détecter le moment le plus opportun pour changer de représentation. La connaissance du temps de couplage théorique de la chaîne de Markov pourrait nous aider à déterminer ce moment. Une étude théorique pour quantifier la différence entre les temps de couplage de l'ensemble des états de la chaîne de Markov et celui du diagramme pourra être envisagée par la suite.

Les diagrammes valués que l'on a décrit dans la conclusion du chapitre 2 pourront faire l'objet d'un travail visant à étendre encore le domaine d'application des diagrammes. Une étude plus poussée du cas des valuations non-discrètes permettrait d'approfondir le sujet de la simulation d'une chaîne de Markov à horizon fixé. Une telle étude contribuerait de plus à l'enrichissement du *package* `DiagramS`, qui ne traite pour l'instant que le cas des diagrammes représentant des ensembles discrets.

Enfin, la parallélisation des algorithmes est une piste que je souhaite explorer afin d'améliorer la complexité des algorithmes du *package* `DiagramS`. En effet, un diagramme est toujours traité colonne par colonne alors que les arcs d'une même colonne sont traités de manière indépendante. Ainsi les algorithmes agissant sur les diagrammes semblent pouvoir se prêter à la programmation concurrente et notamment au paradigme de programmation *map-reduce* [42].

# Bibliographie


[1] S. Asmussen and P. W. Glynn. *Stochastic simulation : algorithms and analysis*, volume 57 of *Stochastic Modelling and Applied Probability*. Springer, New York, 2007.

[2] S. Balsamo. Queueing Networks with Blocking : Analysis, Solution Algorithms and Properties. In D. D. Kouvatsos, editor, *Network Performance Engineering*, volume 5233 of *LNCS*, pages 233–257. Springer Berlin Heidelberg, 2011.

[3] S. Balsamo, A. Marin, and I. Stojic. Perfect Sampling in Stochastic Petri Nets Using Decision Diagrams. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2015 IEEE 23rd International Symposium on*, pages 126–135, Oct 2015.

[4] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *J. Assoc. Comput. Mach.*, 22 :248–260, 1975.

[5] B. Baynat. *Théorie des files d'attente des chaînes de Markov aux réseaux à forme produit*. Hermès - Lavoisier, 2000.

[6] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

[7] G. Birkhoff. *Lattice Theory*, volume 25 of *Colloquium Publications*. American Mathematical Society, 1991. Reprint.

[8] O. Bodini and Y. Ponty. Multi-dimensional Boltzmann Sampling of Languages. In *21st International Meeting on Probabilistic, Combinatorial, and Asymptotic Methods in the Analysis of Algorithms (AofA'10)* , number 01 in AM, pages 49–64, Vienne, Austria, June 2010. 12pp.

[9] A. Bouillard, A. Bušić, and C. Rovetta. Clones : CLOsed queueing Networks Exact Sampling. In *8th International Conference on Performance Evaluation Methodologies and Tools, VALUETOOLS 2014*. ICST, 2014.

[10] A. Bouillard, A. Bušić, and C. Rovetta. Perfect sampling for closed queueing networks. *Performance Evaluation*, 79(0) :146–159, 2014.

[11] A. Bouillard, A. Bušić, and C. Rovetta. Clones : CLOsed queueing Networks Exact Sampling. *Performance Evaluation*, 2015.

[12] A. Bouillard, A. Bušić, and C. Rovetta. Perfect sampling for multiclass closed queueing networks. In *12th International Conference on Quantitative Evaluation of SysTems, QEST 2015*, 2015.

[13] A. Bouillard, A. Bušić, and C. Rovetta. Low complexity state space representation and algorithms for closed queueing networks exact sampling. *Performance Evaluation*, 103 :2–22, 2016.

[14] A. Bouillard and B. Gaujal. Backward Coupling in Bounded Free-Choice Nets Under Markovian and Non-Markovian Assumptions. *Discrete Event Dynamic Systems*, 18(4) :473–498, 2008.

[15] P. Brémaud. *Markov chains : Gibbs fields, Monte Carlo simulation and queues*. Texts in applied mathematics. Springer, New York, Berlin, Heidelberg, 1999.

[16] A. Bušić, S. Durand, B. Gaujal, and F. Perronnin. Perfect sampling of Jackson queueing networks. *Queueing Systems*, 80(3) :37, 2015.

[17] A. Bušić, B. Gaujal, and F. Perronnin. Perfect Sampling of Networks with Finite and Infinite Capacity Queues. In *19th International Conference on Analytical and Stochastic Modeling Techniques and Applications, ASMTA 2012*, volume 7314 of *LNCS*, pages 136–149. Springer, 2012.

[18] A. Bušić, B. Gaujal, and J.-M. Vincent. Perfect Simulation and Non-monotone Markovian Systems. In *3rd International Conference Valuetools'08*, Athens, Greece, 2008. ICST.

[19] J. Buzen. Computational algorithms for closed queueing networks with exponential servers. *Comm. ACM*, 16 :527–531, 1973.

[20] G. Casella and E. I. George. Explaining the Gibbs Sampler. *The American Statistician*, 46(3) :167–174, 1992.

[21] R. B. Cooper. *Introduction to Queueing Theory*. North-Holland, New York, NY, second edition, 1981.

[22] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[23] J. Desel and J. Esparza. *Free Choice Petri Nets*. Cambridge University Press, New York, NY, USA, 1995.

[24] P. Duchon, P. Flajolet, G. Louchard, and G. Schaeffer. Boltzmann samplers for the random generation of combinatorial structures. *Combin. Probab. Comput.*, 13(4-5) :577–625, 2004.

[25] J. Dutka. The early history of the factorial function. 43(3) :225–249, Sept. 1991.

[26] P. Flajolet, É. Fusy, and C. Pivoteau. Boltzmann sampling of unlabelled structures. In *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments and the Fourth Workshop on Analytic Algorithmics and Combinatorics*, pages 201–211. SIAM, Philadelphia, PA, 2007.

[27] P. Flajolet and R. Sedgewick. *Analytic combinatorics*. Cambridge University Press, Cambridge, 2009.

[28] P. Flajolet, P. Zimmermann, and B. V. Cutsem. A calculus for the random generation of labelled combinatorial structures. *Theoretical Computer Science*, 132(1) :1–35, 1994.

[29] Fourneau, J.M. Closed G-networks with Resets : product form solution. *Quantitative Evaluation of Systems, International Conference on*, 00(undefined) :287–296, 2007.

[30] B. Gaujal, G. Gorgo, and J.-M. Vincent. *Perfect Sampling of Phase-Type Servers Using Bounding Envelopes*, pages 189–203. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[31] E. Gelenbe. G-networks : a unifying model for neural and queueing networks. *Ann. Oper. Res.*, 48(1-4) :433–461, 1994.

[32] D. V. Gokhale and S. Kullback. *The information in contingency tables*. Marcel Dekker Inc, New York, 1978.

[33] W. Gordon and G. Newell. Closed queueing systems with exponential servers. *Oper. Res.*, 15,2 :254–265, 1967.

[34] O. Häggström. *Finite Markov chains and algorithmic applications*. London Mathematical Society student texts. Cambridge University Press, Cambridge, New York, Madrid, 2002. Ed. brochée 2003.

[35] G. H. Hardy and S. Ramanujan. Asymptotic Formulaae in Combinatory Analysis. *Proc. London Math. Soc.*, S2-17(1) :75, 1918.

[36] M. Held and R. M. Karp. A dynamic programming approach to sequencing problems. *J. Soc. Indust. Appl. Math.*, 10 :196–210, 1962.

[37] M. Huber. Perfect sampling using bounding chains. *Ann Appl Probab*, 14(2) :734–753, 2004.

[38] J. Jackson. Jobshop-like queueing systems. *Management Sci*, 10,1 :131–142, 1963.

[39] W. Kendall and J. Møller. Perfect simulation using dominating processes on ordered spaces, with application to locally stable point processes. *Advances in Applied Probability*, 32(3) :844–865, 9 2000.

[40] W. S. Kendall. Notes on perfect simulation. In *Markov chain Monte Carlo*, volume 7 of *Lect. Notes Ser. Inst. Math. Sci. Natl. Univ. Singap.*, pages 93–146. World Sci. Publ., Hackensack, NJ, 2005.

[41] S. Kijima and T. Matsui. Randomized approximation scheme and perfect sampler for closed Jackson networks with multiple servers. *Annals of Operations Research*, 162(1) :35–55, 2008.

[42] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon. Parallel Data Processing with MapReduce : A Survey. *SIGMOD Rec.*, 40(4) :11–20, Jan. 2012.

[43] D. A. Levin, Y. Peres, and E. L. Wilmer. *Markov chains and mixing times*. Providence, R.I. American Mathematical Society, 2009. With a chapter on coupling from the past by James G. Propp and David B. Wilson.

[44] R. Marie. An Approximate Analytical Method for General Queueing Networks. *IEEE Transactions on Software Engineering*, 5(5) :530–538, 1979.

[45] A. Nijenhuis and H. S. Wilf. *Combinatorial algorithms*. Academic Press, Inc. [Harcourt Brace Jovanovich, Publishers], New York-London, second edition, 1978. For computers and calculators, Computer Science and Applied Mathematics.

[46] K. N. Olle Häggström. On Exact Simulation of Markov Random Fields Using Coupling from the Past. *Scandinavian Journal of Statistics*, 26(3) :395–411, 1999.

[47] R. O. Onvural. Survey of Closed Queueing Networks with Blocking. *ACM Comput. Surv.*, 22(2) :83–121, 1990.

[48] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking : Bringing Order to the Web, 1999.

[49] J. L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.

[50] C. Pivoteau, B. Salvy, and M. Soria. Boltzmann Oracle for Combinatorial Systems. In *Algorithms, Trees, Combinatorics and Probabilities*, pages 475 – 488. Discrete Mathematics and Theoretical Computer Science, 2008. Proceedings of the Fifth Colloquium on Mathematics and Computer Science. Blaubeuren, Germany. September 22-26, 2008.

[51] J. G. Propp and D. B. Wilson. Exact Sampling with Coupled Markov Chains and Applications to Statistical Mechanics. *Random Struct. Algorithms*, 9(1-2) :223–252, 1996.

[52] J.-L. Rémy. Un procédé itératif de dénombrement d'arbres binaires et son application à leur génération aléatoire. *RAIRO - Theoretical Informatics and Applications - Informatique Théorique et Applications*, 19(2) :179–195, 1985.

[53] C. Rovetta. M-Clones : Multiclass CLOsed queueing Networks Exact Sampling. In *17eme congré annuel de la société Française de Recherche Opérationnelle et d'Aide à la Décision ROADEF 2016*, 2016.

[54] K. Satyam, A. Krishnamurthy, and M. Kamath. Solving general multi-class closed queuing networks using parametric decomposition. *Computers & Operations Research*, 40 :1777–1789, 2013.

[55] K. Sigman. Exact simulation of the stationary distribution of the FIFO M/G/c queue : the general case for $\rho < c$. *Queueing Systems*, 70(1) :37–43, 2012.

[56] J.-M. Vincent. Perfect Simulation of Monotone Systems for Rare Event Probability Estimation. In *Proceedings of the 37th Conference on Winter Simulation*, WSC '05, pages 528–537. Winter Simulation Conference, 2005.

[57] J.-M. Vincent. Perfect simulation, monotonicity and finite queueing networks. In *QEST*, Saint-Malo, 2008.

[58] D. B. Wilson. How to Couple from the Past Using a Read-once Source of Randomness. *Random Struct. Algorithms*, 16(1) :85–113, Jan. 2000.

## Résumé

La génération aléatoire d'objets combinatoires est un problème qui se pose dans de nombreux domaines de recherche (réseaux de communications, physique statistique, informatique théorique, combinatoire, etc.). Couramment, la distribution des échantillons est définie comme la distribution stationnaire d'une chaîne de Markov ergodique. En 1996, Propp et Wilson ont proposé un algorithme permettant l'échantillonnage sans biais de la distribution stationnaire. Ce dernier appelé aussi algorithme de simulation parfaite, requiert la simulation en parallèle de tous les états possibles de la chaîne. Plusieurs stratégies ont été mises en œuvre afin de ne pas avoir à simuler toutes les trajectoires. Elles sont intrinsèquement liées à la structure de la chaîne considérée et reposent essentiellement sur la propriété de monotonie, la construction de processus bornants qui exploitent la structure de treillis de l'espace d'états ou le caractère local des transitions.

Dans le domaine des réseaux de communications, on s'intéresse aux performances des réseaux de files d'attente. Ces derniers se distinguent en deux groupes : ceux dont la distribution stationnaire possède une forme produit qui est facile à évaluer par le calcul et les autres. Pour ce dernier groupe, on utilise la génération aléatoire pour l'évaluation de performances. De par la structure des chaînes qui leurs sont associées, les réseaux ouverts de files d'attente se prêtent bien à la simulation via l'algorithme de simulation parfaite mais pas les réseaux fermés. La difficulté réside dans la taille de l'espace des états qui est exponentielle en le nombre de files à laquelle s'ajoute une contrainte globale à savoir le nombre constant de clients.

La contribution principale de cette thèse est une nouvelle structure de données appelée *diagramme*. Cette structure est inspirée de la programmation dynamique et introduit une nouvelle technique de construction de processus bornant. La première partie du manuscrit est consacrée à la mise en œuvre de l'algorithme de Propp et Wilson pour des réseaux fermés n'étant pas nécessairement à forme produit. La représentation des états par un diagramme et l'opération de transition pour le processus bornant a dès lors une complexité polynomiale en le nombre de files et de clients. Cette technique est ensuite étendue aux réseaux fermés multiclasses ainsi qu'aux réseaux possédant des synchronisations. Une spécification des ensembles d'objets pouvant être représentés par un diagramme ainsi que des algorithmes agissant sur cette structure de données sont également proposés dans cette thèse.

La méthode de Botzmann est une autre technique de simulation sans biais. Basée sur la combinatoire analytique, elle permet l'échantillonnage uniforme d'objets appartenant à une même classe combinatoire. Elle est employée dans la seconde partie de cette thèse afin d'échantillonner la distribution stationnaire de réseaux fermés à forme produit et pour la génération des multi-ensembles de taille fixe. Dans ce cadre, les diagrammes sont une nouvelle fois mis à profit. Enfin, la troisième partie présente les logiciels découlant des travaux présentés tout au long de ce travail, et qui implémentent les diagrammes et mettent en œuvre la simulation parfaite de réseaux fermés de files d'attente.



## Mots Clés

Chaîne de Markov, Simulation, Théorie des files d'attentes, Simulation parfaite, Générateur de Boltzmann



## Abstract

Random generation of combinatorial objects is an important problem in many fields of research (communications networks, theoretical computing, combinatorics, statistical physics, ...). This often requires sampling the stationary distribution of an ergodic Markov chain. In 1996, Propp and Wilson introduced an algorithm to produce unbiased samples of the stationary distribution, also called a perfect sampling algorithm. It requires parallel simulation of all possible states of the chain. To avoid simulating all the trajectories, several strategies have been implemented. But they are related to the structure of the chain and require a monotonicity property, or a construction of a bounding chain that exploits the lattice structure of the state space or the local character of the transitions.

In the field of communications networks, attention is paid to the performance of queueing networks, that can be distinguished into two groups: the networks that have a product form stationary distribution which is easy to compute. Random generation can be used for the others. Perfect sampling algorithms can be used for open queueing networks, thanks to the lattice structure of their state space. Unfortunately, that is not the case for closed queueing networks, due to the size of the state space which is exponential in the number of queues and a global constraint (a constant number of customers).

The main contribution of this thesis is a new data structure called *diagram*. It is inspired by dynamic programming and allows a new technique of construction of bounding processes. The first part of the manuscript is devoted to the implementation of the Propp and Wilson algorithm for closed queueing networks. The representation of a set of states by a diagram and the transition operation for the bounding process has a polynomial complexity in the number of queues and customers. This technique is extended to closed multi-class networks and to networks with synchronizations. Specification of sets of objects that can be represented by a diagram and generic algorithms that use this data structure are proposed in this manuscript.

The Botzmann method is another unbiased sampling technique. It is based on analytical combinatorics and produces uniform samples from objects that belong to the same combinatorial class. It is used in the second part of this thesis in order to sample the stationary distribution of closed networks with product form and for the generation of multisets of fixed cardinality. Diagrams are used again in this context. Finally, the third part presents the software produced during this thesis, implementing diagrams and perfect simulation of closed queueing networks.



## Keywords

Markov chains, Simulation, Queueing theory, Perfect sampling, Boltzmann sampling