Prédiction d'alarmes dans les réseaux via la recherche de motifs spatio-temporels et l'apprentissage automatique

Achille Salaün

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2021IPPAS010

Thèse de doctorat

NOKIA Bell Labs

TELECOM SudParis

IP PARIS

Prédiction d'alarmes dans les réseaux via la recherche de motifs spatio-temporels et l'apprentissage automatique

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom SudParis

École doctorale n°574 Ecole Doctorale de Mathématiques Hadamard (EDMH)
Spécialité de doctorat : Mathématiques aux interfaces

Thèse présentée et soutenue à Paris, le 8 juillet 2021, par

ACHILLE SALAÜN

Composition du Jury :

Erwan Le Pennec Professeur, École Polytechnique (CMAP)	Président
Maurizio Filippone Maître de conférence HDR, EURECOM	Rapporteur
Matthieu Latapy Directeur de recherche, CNRS (LIP6)	Rapporteur
Johanne Cohen Directrice de Recherche, CNRS (LISN)	Examinatrice
François Desbouvries Professeur, Télécom SudParis (SAMOVAR)	Directeur de thèse
Anne Bouillard Ingénieure de recherche, Nokia Bell Labs	Co-directrice de thèse
Marc-Olivier Buob Ingénieur de recherche, Nokia Bell Labs	Invité
Yohan Petetin Maître de conférence, Télécom SudParis (SAMOVAR)	Invité

Remerciements

Je tiens tout d'abord à remercier mes encadrants de thèse, François Desbouvries, Yohan Petetin, Anne Bouillard et Marc-Olivier Buob, pour leur bienveillance, leur patience et leur appui indéfectible. J'ai énormément appris à vos côtés, tant sur le plan scientifique que sur le plan humain et ne saurais suffisamment vous exprimer ma gratitude.

J'aimerais remercier chaleureusement Maurizio Filippone et Matthieu Latapy pour la relecture de mon manuscrit.

Je souhaiterais également remercier mes collègues, à Nokia Bell Labs comme au LINCS, qui ont donné à mon travail un cadre fantastique pendant ces trois années. Je voudrais remercier tout particulièrement Élie De Panafieu, François Durand et Fabien Mathieu qui ont accepté de faire partie de mon « jury » le temps d'une répétition générale et m'ont ainsi apporté une aide et des conseils précieux. Par ailleurs, un grand merci à Kamélia Daudel, camarade d'écoles et amie, qui a su, entre deux articles de revue, m'initier aux *webtoons* et à l'échantillonnage d'importance.

Enfin, j'aimerais remercier mes proches pour leurs soins et leur optimisme constant. En particulier, merci à mes parents et ma sœur qui ont été mes *supporters* de la première heure et sont ceux de chaque instant.

Cette thèse s'est finie dans un contexte de crise sanitaire mondiale : personne n'a été épargné par l'isolement et le stress induits par la situation ; pourtant, vous m'avez soutenu tous autant que vous êtes, malgré ces difficultés. Cela n'en est que plus remarquable et je me devais de le rappeler.

Du fond du cœur, merci.

aux docteurs Pierre et Antoine Salaün, à Geneviève Salaün
à Heinz, Marie et Bruno Riedel.

Table des matières

Remerciements **3**

Introduction **11**

Les réseaux d'aujourd'hui 11

Les problèmes en gestion de réseau 13

Plan de thèse et contributions 14

Acronymes et notations 18

1 Outils et concepts pour la gestion de réseaux **20**

1.1 Données pour la gestion de réseaux 20

1.1.1 Mesures actives et mesures passives 20

1.1.2 Analyse de logs 22

1.1.3 Représenter un log d'alarmes 24

1.2 Concepts clés pour le diagnostic de fautes 25

1.2.1 Quelques mots sur le concept de causalité 25

1.2.2 Interprétabilité : comprendre le résultat d'un algorithme 27

1.3 Analyse de causes racines 28

1.3.1 Techniques basées sur l'analyse de mots 28

1.3.2 Méthodes par apprentissage 35

2 Modèles pour la prédiction de données temporelles 39

2.1 Introduction à la prédiction de séries temporelles 39

2.1.1 Prédiction dans le cas d'une loi jointe connue 39

2.1.2 Prédiction lorsque la loi jointe n'est pas connue 41

2.1.3 Prédiction de séries temporelles 43

2.2 Modèles markoviens 44

2.2.1 Chaînes de Markov 45

2.2.2 Chaînes de Markov cachées 45

2.3 Architectures neuronales 56

2.3.1 Introduction aux réseaux de neurones 56

2.3.2 Réseaux de neurones récurrents 57

2.4 Conclusion 60

3 Stockage des motifs de causalité observés dans un log 61

3.1 Représentation d'un log d'alarmes 62

3.1.1 Collection d'événements 63

3.1.2 Causalités spatio-temporelles 64

3.1.3 Deux façons de représenter un log 66

3.1.4 Langage des mots à stocker 69

3.2 Construction d'une structure de stockage 72

3.2.1 Définition formelle d'un DIG-DAG 72
3.2.2 Algorithme de stockage naïf et DIG-trie 73
3.2.3 Algorithme de stockage optimisé et DIG-DAG compact 76
3.2.4 Pondération des arcs 81
3.3 Variantes 82
3.3.1 Minimalité 82
3.3.2 Étiquetage des transitions par des mots 84
3.4 Conclusion 84

4 Exploitation du DIG-DAG **86**

4.1 Extraction de motifs pertinents 87
4.1.1 Requêtes 87
4.1.2 Requêtes régulières 88
4.2 Simplification d'un sous-DIG-DAG 92
4.2.1 Réduction transitive 94
4.2.2 Minimisation 98
4.2.3 Simplification des racines d'un sous-DIG-DAG 98
4.3 Conclusion 99

5 Applications expérimentales **100**

5.1 Méthodologie pour l'analyse de bout en bout d'un log d'alarmes 101
5.1.1 Chargement du log 101
5.1.2 Construction du DIG-DAG 103

5.1.3 Requêtes sur le DIG-DAG 104
5.1.4 Conseils pour les logs de grande taille 106
5.2 Évaluations expérimentales 111
5.2.1 Passage à l'échelle 111
5.2.2 Précision du diagnostic de fautes 114
5.3 Exemples d'analyses de logs 116
5.3.1 Simplification de l'alphabet et construction du DIG-DAG 117
5.3.2 Utilisation de la topologie 118
5.3.3 Requêtes sur le DIG-DAG 119
5.3.4 Pertinence des requêtes 119
5.4 Conclusion 121

6 Expressivité comparée des modèles HMM et RNN 123

6.1 HMM, RNN et modèles génératifs unifiés (GUM) 126
6.2 GUM linéaires et gaussiens : le cas unidimensionnel 128
6.2.1 Calculs préliminaires 129
6.2.2 Matrices de covariance Toeplitz géométriques 131
6.2.3 Classes d'équivalences 134
6.2.4 Expressivité des RNN et HMM 136
6.3 Généralisation au cas multidimensionnel 139
6.3.1 Stationnarité 140
6.3.2 Image réciproque de la fonction de covariance et théorie de la réalisation stochastique 142

6.3.3 Expressivité des GUM, HMM et RNN 146

6.3.4 Lien avec le cas unidimensionnel 153

6.4 Conclusion et perspectives 156

Conclusion et perspectives **159**

A Introduction à la théorie de la réalisation **163**

A.1 Théorie de la réalisation déterministe 163

A.2 Théorie de la réalisation stochastique 165

Introduction

Les réseaux d'aujourd'hui

L'avènement et l'évolution des réseaux de télécommunication a marqué une transformation majeure du monde en permettant le partage massif d'informations à haut débit à travers le globe. Au cours des dernières décennies, cette hyperconnectivité s'est profondément ancrée dans notre quotidien ainsi que dans la plupart des domaines d'activité. À titre d'exemple, les GAFAM (Google, Amazon, Facebook, Apple, Microsoft) et leurs analogues chinois BHATX (Baidu, Huawei, Alibaba, Tencent, Xiaomi) se sont imposés en mastodontes économiques. [1] Malgré cette hégémonie, ces entreprises sont particulièrement dépendantes des opérateurs de télécommunication : ainsi, 99% du trafic internet transite par des câbles sous-marins [François and Aubry, 2018]. Plus généralement, il existe une grande variété de réseaux pour une grande variété d'usages. Par exemple, *Open Transit Internet* [oti, 2020] permet d'accéder au réseau internet à travers une structure sécurisée et internationale. La téléphonie mobile est quant à elle supportée par des opérateurs comme Verizon ou Orange. Cloudflare propose un réseau de distribution de contenu permettant aux utilisateurs de mettre en cache des données sur des serveurs distants [clo, 2019]. Renater offre une infrastructure réseau dédiée à la mise en place de projets scientifiques et académiques [ren, 2019]. Quels que soient ces réseaux, il est critique pour les opérateurs de pouvoir assurer une bonne qualité de service selon les critères définis dans l'« entente de niveau de service » (SLA) avec leurs clients. Pour cela, des solutions de diagnostic de fautes et de résolution de

1. Par exemple, le 21 mai 2020, parmi les cent plus grandes entreprises cotées au NASDAQ (*National Association of Securities Dealers Automated Quotations*, l'une des principales bourses américaines avec le *New York Stock Exchange* (NYSE)), les GAFAM font partie des dix entreprises représentant 50% de l'indice. Parmi les cinq autres entreprises de ce top 10, on peut encore citer Cisco, Intel, Adobe ou Comcast qui sont tous des acteurs des télécommunications et du numérique (seule Pepsi se démarque par son domaine d'activité).

pannes sont requises afin de préserver temps, argent et ressources humaines.

Toutefois, il est difficile de gérer de tels réseaux. Tout d'abord, leur taille, en termes de nombre de composants et de liens entre ces composants, est généralement grande. Ces réseaux étant distribués, il est souvent nécessaire de croiser, agréger et synthétiser des informations collectées à plusieurs endroits. De plus, ces réseaux impliquent des technologies variées. On peut par exemple citer la co-existence des technologies radio 2G, 3G et 4G dans les réseaux de téléphonie mobile. De même, dans les réseaux internet, il est possible de rencontrer plusieurs protocoles de routage différents tels que BGP (*Border Gateway Protocol*) [Rekhter et al., 1994], OSPF (*Open Shortest Path First*) [Moy, 1998], LDP (*Label Distribution Protocol*) [Andersson et al., 2007] ou encore RSVP (*Resource Reservation Protocol*) [Braden et al., 1997]. Divers services réseaux comme DNS (*Domain Name System*) [Mockapetris, 1987] ou DHCP (*Dynamic Host Configuration Protocol*) [Droms, 1997] sont employés dans les réseaux locaux. Si les réseaux évoluent selon les usages et les demandes de leurs utilisateurs, certains usages sont encore attachés aux infrastructures plus anciennes. Autrement dit, de nouvelles technologies apparaissent tandis que d'autres, plus âgées (voire en passe d'être obsolètes), sont conservées. Par exemple, les travaux réalisés en 5G mènent à l'introduction de nouveaux composants et protocoles pour aborder de nouvelles façons de connecter personnes, objets et services, tandis que des équipements tels que les ascenseurs ou les compteurs électriques sont souvent rattachés au réseau 2G. De fait, le démantèlement de ce dernier ne saurait être d'actualité [Gauthier, 2020]. Un enjeu en gestion de réseaux est donc d'harmoniser l'ensemble des informations venant de sources de natures différentes. Notamment, il n'est pas dit qu'une personne suffise à comprendre une panne dans son ensemble. Ainsi, puisque les infrastructures existantes sont déjà complexes à gérer (la preuve en est la profusion d'outils qui aident à surveiller leur état) et que les variétés et les nombres d'équipements, de technologies et d'usages vont continuer à croître à cause d'une place toujours plus importante du numérique dans notre société, analyser un réseau va être une tâche de plus en difficile.

Les réseaux de télécommunication sont donc des systèmes complexes et sujets à l'erreur. Ces erreurs peuvent alors être nombreuses, provenir de causes variées et prenant des degrés de sévérité allant de l'erreur récurrente et anecdotique à la panne du système entier. Par ailleurs, un réseau est un système dynamique évoluant en temps réel. Une erreur pouvant en causer une autre, connaître la chronologie des erreurs observées (c'est-à-dire savoir dans quel ordre, voire quand elles sont apparues) permet de mieux comprendre l'état du système. Cependant, saisir cette composante temporelle est également difficile : par exemple, des

erreurs d'origines différentes peuvent avoir lieu simultanément.

Les problèmes en gestion de réseau

Nous pouvons distinguer en gestion de réseau trois grands problèmes.

Détection d'anomalies

Puisque les réseaux sont sujets aux erreurs, il est important d'être en mesure de les détecter rapidement. En effet, une *détection d'anomalies* efficace est nécessaire pour commencer le traitement de l'erreur le plus tôt possible et donc de limiter le temps pendant lequel la qualité de service est dégradée. L'aspect temporel est particulièrement critique et l'état du système doit pouvoir être traité en ligne.

Analyse de causes racines

Une fois une erreur repérée, l'opérateur doit pouvoir comprendre comment elle a été provoquée et quelle en est l'origine. Ainsi, après avoir identifié la cause d'une panne grâce à l'*analyse de causes racines*, ce dernier peut plus facilement apporter la correction appropriée. Ici, il n'est pas nécessaire de traiter l'état du système en temps réel puisqu'il s'agit de l'analyse *a posteriori* d'une panne.

Prédiction de pannes

À terme, il serait idéal de pouvoir corriger une panne avant même qu'elle ait lieu, et pour cela faire de la *prédiction de pannes*. Autrement dit, le crédo de la prédiction de pannes est de prévenir plutôt que de guérir. Comme la détection d'anomalies, l'aspect temporel est important puisque la prédiction doit être réalisée le plus en amont possible de l'occurrence de la panne et ainsi laisser le temps de maintenir le système pour que la panne n'ait jamais lieu. Pour que la prédiction soit la plus précise possible, une telle solution doit donc intégrer les observations les plus récentes concernant l'état du système en fonctionnant en ligne. En général, un incident implique un grand nombre d'erreurs, de sévérités, de types

et de fréquences d'occurrence variés. Ainsi, prédire uniquement la prochaine erreur n'est pas suffisant pour déterminer si celle-ci va conduire à une panne critique. À l'inverse, être capable de prédire si une telle panne est probable et de caractériser cette panne est beaucoup plus pertinent sur le plan pratique.

De manière générale, parmi l'ensemble des erreurs observées dans l'historique du système, une notion de causalité doit être définie afin de déterminer comment les erreurs observées s'articulent les unes par rapport aux autres. Dans le cas de l'analyse de causes racines, il s'agit de remonter à la source de ces cascades d'erreurs, tandis que la prédiction cherche à en décrire l'évolution.

Plan de thèse et contributions

Si la détection d'anomalies est un sujet bien traité, l'analyse de causes racines et plus encore la prédiction de pannes restent des sujets ouverts. Dans l'absolu, notre objectif est de traiter le problème de prédiction de pannes dans les réseaux et, à défaut, celui de l'analyse de causes racines. Pour cela, nous proposons une structure stockant les motifs de causalité observés dans l'historique d'un réseau. Cette structure et sa construction ont été pensées pour la prédiction de panne et l'analyse de causes racines. En particulier, nous détaillons comment cette structure peut être utilisée dans un contexte d'analyse de causes racines en recherchant des motifs spécifiques à l'aide d'un outil de requêtes. Enfin, nous présentons une comparaison théorique du pouvoir modélisant de deux modèles classiques pour la prédiction de données temporelles : les chaînes de Markov cachées et les réseaux de neurones récurrents. Cette comparaison repose notamment sur la définition d'un modèle englobant ces deux modèles sous un point de vue génératif.

Plus précisément, dans le chapitre 1, nous introduisons différents éléments clés pour la gestion d'un réseau. Tout d'abord, nous présentons les techniques actuelles de surveillance de réseaux et la nature des données qu'elles mettent à disposition. Nous présentons ensuite deux concepts que nous estimons importants à intégrer dans de telles solutions. D'une part, il s'agit de prendre en compte les interactions entre machines, régies par le concept de causalité. D'autre part, l'opérateur en charge doit pouvoir comprendre les résultats obtenus, ce qui est décrit par la notion d'interprétabilité. Enfin, un état de l'art des solutions d'analyse de causes racines est fourni.

Dans un second temps, le chapitre 2 présente plusieurs modèles pour la prédiction de données temporelles. Ces modèles sont issus de deux familles différentes : les réseaux bayésiens et les réseaux de neurones artificiels. En particulier, deux modèles se distinguent : chaînes de Markov cachées et réseaux de neurones récurrents. Malgré leurs différences, ils peuvent être vus comme deux types de modèles génératifs aux structures de dépendances similaires.

Dans le chapitre 3, nous proposons une généralisation de l'algorithme d'Ukkonen pour stocker de manière interprétable dans une structure de graphe tous les motifs observés de causalité potentielle dans un log. Cette structure est appelée DIG-DAG et sa construction est faite en ligne en tenant compte des connaissances *a priori* de l'expert sur le système.

Afin d'exploiter cette structure dans un contexte d'analyse de causes racines, le chapitre 4 présente un ensemble d'outils permettant d'isoler hors ligne des motifs arbitraires au sein d'un DIG-DAG.

Le chapitre 5 présente alors une méthodologie pour l'analyse de causes racines, de l'ingestion du log à l'extraction de motifs de causalité pertinents via la construction d'un DIG-DAG. Des résultats expérimentaux sont également présentés afin de juger les performances de notre solution.

Finalement, nous étudions le problème de la prédiction sous un autre angle : celui des modèles génératifs. Plus particulièrement, nous cherchons dans le chapitre 6 à comparer l'expressivité [2] des modèles de Markov cachés à celle des réseaux de neurones récurrents. Pour cela, nous définissons un modèle (GUM pour *Generative Unified Model*) qui englobe HMM et RNN sous un point de vue génératif. Dans le cas particulier linéaire gaussien, nous écrivons analytiquement l'expressivité de ce modèle.

2. c'est-à-dire la capacité d'un modèle à modéliser une grande variété de distributions

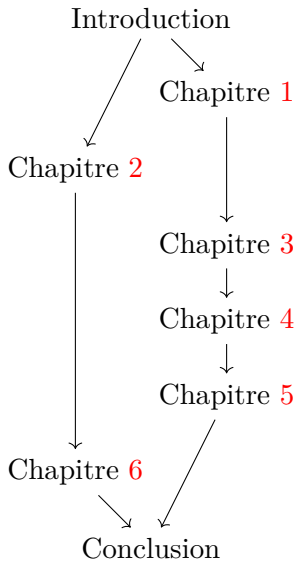


FIGURE 1 – Structure de dépendances des chapitres de cette thèse ; chaque arc indique l'utilisation de notions du chapitre source par le chapitre cible.

Liste des contributions

Cette thèse a mené aux contributions suivantes :

- Une nouvelle structure causale appelée DIG-DAG, stockant les motifs de causalité potentielle observés dans un flux d'événements, ainsi qu'un algorithme de construction en ligne de cette structure [Bouillard et al., 2018]
  Log analysis via space-time pattern matching (CNSM 2018), Bouillard A., Buob M.-O., Raynal M. et Salaün A.
- Un système de requêtes dédié à l'extraction de motifs dans une structure causale, ainsi qu'une méthodologie de bout en bout pour l'analyse de causes racines dans un log d'alarmes [Salaün et al., 2019a]
  Space-time pattern extraction in alarm logs for network diagnosis (MLN 2019), Salaün A., Bouillard A. et Buob M.-O.
- Implémentation de l'algorithme de construction du DIG-DAG et des outils de requêtes en Python 3. Le code a notamment servi de prototype à des *business units* de Nokia
- Présentation de nos outils et méthodologies pour l'analyse de causes racines dans les logs [Salaün et al., 2020]

Demo abstract : End-to-end root cause analysis of a mobile network (INFOCOM WKSHPS 2020), Salaün A., Bouillard A. et Buob, M.-O.

- Un modèle graphique unifiant HMM et RNN sous un point de vue génératif, ainsi qu'une comparaison théorique de l'expressivité de ces deux modèles [Salaün et al., 2019b]
  Comparing the modeling powers of RNN and HMM (ICMLA 2019), Salaün A., Petetin Y. et Desbouvries F.

Voici les correspondances entre les contributions précédentes et les chapitres de ce manuscrit :

Contribution	Chapitre(s)
CNSM 2018	3
MLN 2019	4, 5
INFOCOM 2020	5
ICMLA 2019	6

Acronymes et notations

Dans cette thèse, nous utiliserons les acronymes suivants :

HMM	*Hidden Markov Model*
EM	Algorithme *Expectation-Maximization*
RNN	*Recurrent Neural Network*
GRU	*Gated Recurent Unit*
LSTM	*Long Short Term Memory*
GUM	*Generative Unified Model*
DIG	*Directed Interval Graph*
DIG-DAG	*Directed Interval Graph - Directed Acyclic Graph*

Par ailleurs, dans les chapitres 2 et 6, nous utilisons les notations suivantes :

x_t	Observation à l'instant t
$x_{0:t}$	Suite d'observations entre les instants 0 et t
h_t	Variable latente à l'instant t
$h_{0:t}$	Suite de variables latentes entre les instants 0 et t

Notons que tant que cela ne posera pas d'ambiguïté, nous confondrons variables aléatoires et réalisations. De même, selon le contexte, la notation $p(.)$ peut désigner une probabilité ou une densité.

Pour rappel, l'expression de la densité de probabilité de la variable aléatoire normale de moyenne μ et de variance σ^2 est pour tout $x \in \mathbb{R}$:

$$\mathcal{N}(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Les notations qui suivent sont quant à elles employées dans les chapitres 3, 4 et 5.

Σ	Ensemble fini, alphabet
Σ^*	Langage des mots sur Σ
$\sigma \in \Sigma$	Symbole
s, t	Dates (resp. de début et de fin)
$e = (\sigma, [s, t])$	Événement décrivant l'occurrence de $\sigma \in \Sigma$ entre les dates s et t
$\mathcal{E} = (e_i)_{i \in [\![1;n]\!]}$	Log d'alarmes
$\mathcal{D}$	DIG-DAG
λ	Fonction d'étiquetage d'un DIG-DAG
$\mathcal{A}$	Ensemble des sommets actifs d'un DIG-DAG
$u, v \in V$	sommets d'un DIG-DAG
u_0	Racine d'un DIG-DAG
$g(u, \sigma)$	Sommet cible de la transition caractérisée par (u, σ)
$n_{u,\sigma}$	Nombre d'activations de l'arc (u, σ)
$r_{u,\sigma}$	Ratio de l'arc (u, σ)
ρ	Seuil sur les ratios

Chapitre 1

Outils et concepts pour la gestion de réseaux

Ce chapitre est une introduction à la gestion de réseaux. En particulier, dans la section 1.1, nous expliquons comment les réseaux sont surveillés et quelles sont les données issues de cette surveillance. Ces données sont généralement consignées dans des logs dont nous présentons quelques solutions d'analyse présentes sur le marché. Nos travaux reposent ainsi sur l'étude de ces logs. Dans un second temps, nous introduisons en section 1.2 deux concepts que nous jugeons clés pour toute solution de diagnostic de fautes. Tout d'abord, nous considérons le concept de causalité, qui décrit le lien entre les éléments d'un log. Ensuite, nous considérons celui d'interprétabilité, c'est-à-dire la capacité d'un outil à fournir des résultats compréhensibles par un opérateur humain chargé du système. Enfin, nous proposons un état de l'art des solutions d'analyse de causes racines en section 1.3.

1.1 Données pour la gestion de réseaux

1.1.1 Mesures actives et mesures passives

La gestion de réseau passe par la collecte d'information auprès des composants logiciels et matériels impliqués, réalisée au travers de mesures. Ces mesures peuvent être actives (déclenchées par un moniteur) ou passives (stockées par le programme lui-même).

Une mesure active est une mesure qui est déclenchée arbitrairement. Ainsi, les *indicateurs clés de performance* (KPI pour *Key Performance Indicator*) correspondent à des mesures programmées à intervalles réguliers. Ils permettent un suivi général de l'état de santé du système. Les mesures réalisées peuvent être variées. Elles peuvent concerner directement les équipements présents dans le réseau. Par exemple, il peut s'agir de mesurer la température d'une machine ou l'utilisation de sa mémoire vive. Des *battements de cœur* peuvent également être planifié pour vérifier régulièrement qu'une machine donnée est active. Il est également possible de surveiller l'état des connexions entre les machines du réseau. Par exemple, l'outil `ping` (pour *Packet INternet Groper*) envoie à une machine ciblée une demande d'accusé de réception à travers le réseau. Dès la réception du message, la machine ciblée retourne l'accusé à la machine source, ce qui permet de savoir si la machine ciblée est accessible et quel est le temps nécessaire à un paquet pour transiter d'une machine à l'autre. L'outil `traceroute` retrace quant à lui le chemin pris par un paquet entre deux machines. Cela permet de connaître la liste des routeurs traversés, le temps pris par un paquet pour passer d'un routeur à un autre ainsi que la quantité de paquets perdus pendant la mesure.

Contrairement aux mesures actives, les mesures passives visent à écouter et recenser les événements qui ont lieu dans le réseau. Ces événements sont alors mémorisés dans des fichiers appelés *logs d'alarmes* ou *fichiers de trace*. Par exemple, les outils *Wireshark* [wir, 2021] ou `tcpdump` [tcp, 2021] permettent de capturer les paquets lorsque ces derniers transitent dans le réseau. Une mesure passive peut aussi être créée à partir de mesures actives. Ainsi, un événement peut être défini à partir d'un KPI en le seuillant : par exemple, si la température d'une machine dépasse un seuil critique, une alerte est alors émise.

Des solutions comme *Cacti* [cac, 2020], *Munin* [mun, 2020] ou *Nagios* [nag, 2020] supervisent et ordonnancent la surveillance d'un réseau. Pour cela, ces outils permettent de visualiser l'état du système à l'aide d'un tableau de bord regroupant un ensemble de mesures actives et/ou passives. Parmi les quantités mesurées, on trouve la température d'une machine (en passant par un capteur), l'utilisation d'un processeur, la latence réseau ou encore la qualité de service d'une ligne mais aussi des notifications lorsque certaines mesures dépassent un seuil d'alerte.

Enfin, pour un même réseau, les anomalies détectées sont généralement reportées dans un fichier appelé log. Plus précisément, chaque machine héberge généralement plusieurs applications et chaque application peut engendrer un ou plusieurs fichiers de logs. Afin d'avoir un point de vue global du système, il existe des aggrégateurs de logs qui centralisent

l'ensemble des logs d'un réseau. Ainsi, `rsyslog` [rsy, 2020] formalise les logs et les mesures d'un système en un même fichier de trace pour en faciliter l'accès et le traitement. Les événements y sont inscrits au format *Syslog* et sont ainsi caractérisés par une date, le nom de la machine émettrice, la nature du processus qui a déclenché l'événement, sa sévérité ainsi qu'un identifiant et un corps de message.

1.1.2 Analyse de logs

Les trois problèmes que sont la détection d'anomalies, l'analyse de causes racines et la prédiction de pannes peuvent être résolus de manière plus spécifique sur des logs. Si tout événement présent dans un log est *a priori* anormal, certains le sont moins que d'autres, voire habituels. Même dans un système sain, des événements sont régulièrement relevés. La détection d'anomalies revient donc à isoler les portions du log les plus critiques. L'analyse de causes racines correspond quant à elle à la transformation d'une suite d'événements dont les liens de causalité sont inconnus *a priori* (un log) en une explication de la panne basée sur ces liens trouvés *a posteriori*. Enfin, la prédiction de pannes se résume à la prédiction d'événements traduisant un comportement anormal du réseau.

L'analyse de logs est cependant difficile car ces fichiers sont verbeux et bruités du fait de la complexité inhérente aux réseaux. La verbosité se traduit notamment par un flux volumineux de données. Ainsi, *Kafka* [kaf, 2017] est une solution générique pour la manipulation de flux de données en temps réel. En particulier, ces flux peuvent avoir un haut débit et *Kafka* permet leur ingestion et leur traitement. Cette solution est pertinente pour la gestion de réseau dans la mesure où un grand nombre de mesures est émis à une fréquence élevée. Des outils comme `grep` [Thompson, 1968] et *ElasticSearch* [ela, 2020] servent à extraire des informations conformes à une recherche étant donné un volume conséquent de données. Dans un contexte d'analyse de logs, ils aident les administrateurs réseaux à rechercher des motifs d'erreurs dans des fichiers de trace volumineux. Plus spécifiquement, `grep` est un outil générique pour l'extraction de motifs dans un fichier texte à partir d'expressions rationnelles. Quant à *ElasticSearch*, il permet d'indexer et fouiller de manière distribuée des données en temps réel parmi des documents de tout type.

Toutefois, filtrer efficacement un flux de données volumineux n'est pas toujours suffisant pour expliquer un problème. Ainsi, les événements recensés ont des sévérités variables allant du simple message informatif à des erreurs graves. La fréquence des événements y est aussi

variable : certains événements sont régulièrement actifs tandis que d'autres sont très localisés dans le temps. Enfin, les interactions entre les machines étant complexes, il est difficile de retracer une explication de la suite des événements observés. Par exemple, plusieurs cascades indépendantes d'événements peuvent avoir lieu au même moment. Pour pallier ces difficultés, il existe des solutions qui intègrent surveillance, détection d'anomalies et outils d'analyse de logs. Nous pouvons par exemple citer *Anodot* [ano, 2020], *Loggly* [log, 2020] ou *Logentries* [log, 2019]. Ces outils centralisent les mesures faites sur le système et permettent de les visualiser de manière interactive. Par ailleurs, ils peuvent également fournir un pré-traitement de ces données grâce à des solutions de détection d'anomalies ou de filtrage des comportements normaux. De plus en plus de ces fonctionnalités reposent sur des techniques d'apprentissage automatique. Certains outils de surveillance et de gestion de réseaux, comme ceux proposés par *PacketAI* [pac, 2020], reposent ainsi principalement sur l'apprentissage automatique.

Caractéristiques pertinentes pour l'analyse de logs

L'analyse d'un log peut s'appuyer sur certaines caractéristiques ou données complémentaires comme le nombre d'occurrence de chaque type d'alarmes dans une fenêtre de temps donnée, l'ordre d'arrivée de ces alarmes ou encore la localisation de ces alarmes.

Dans un premier temps, une façon naïve de comparer des événements entre eux est de considérer leurs fréquences d'apparition respectives. Autrement dit, distinguer un événement fréquent d'un événement rare fournit une première grille de lecture à l'administrateur réseau. Ainsi, selon le contexte, il pourra décider si un événement rare est plutôt l'anomalie caractéristique de la panne ou plutôt un événement anecdotique. Inversement, un événement fréquent peut être considéré comme un élément problématique à corriger de toute urgence ou juste comme du bruit peu utile.

Par ailleurs, il est également pertinent de prendre en compte la chronologie du log. *A priori*, un événement tardif est susceptible d'être un symptôme tandis qu'un événement précoce est davantage susceptible d'être une cause. Ainsi, l'ordre d'arrivée des événements inscrits dans le log revêt une certaine importance dans la mesure où la cause précède l'effet. Cet ordre est induit par défaut par l'ordre d'apparition dans le fichier. Toutefois, les événements de certains logs comportent des dates de début et parfois aussi de fin d'alarmes. Naturellement, un horodatage précis du log simplifie son analyse. Par exemple, dans un sys-

tème distribué, il n'est pas garanti que les horloges des différents *loggers* soit synchronisées, et leur analyse groupée peut dès lors s'en trouver compliquée. Alors, pour deux événements proches, il est possible que leur ordre d'apparition dans le log soit l'inverse de leur ordre de réalisation. Par conséquent, il est important que la temporalité exprimée dans un log corresponde à ce qu'il s'est effectivement passé dans le système. Pour cela, les logs d'un réseau peuvent être aggrégés en temps réel avec des outils comme `rsyslog` ou voir leurs horloges respectives synchronisées en utilisant par exemple NTP (pour *Network Time Protocol*) [ntp, 2014].

Enfin, connaître à l'avance la topologie du réseau permet de mieux comprendre les interactions entre les machines. En effet, la propagation des erreurs à travers le réseau se fait de machine à machine, de programme à programme. Par conséquent, connaître la topologie physique (resp. logique) du réseau permet d'écarter d'office les interactions entre des machines (resp. programmes) indépendantes. Même une connaissance partielle de la topologie peut aider significativement au diagnostic de fautes. Malheureusement, ces connaissances ne sont pas toujours disponibles.

1.1.3 Représenter un log d'alarmes

La façon la plus basique de représenter un log d'alarmes est de le considérer comme une suite chronologique d'événements. La notion d'événements n'est pas propre aux réseaux de télécommunication et peut être retrouvée dans différents contextes [Wang et al., 2020]. Par exemple, l'activité d'un réseau social peut être modélisée comme tel, ce qui permet de suivre l'évolution de la tendance d'un sujet. Néanmoins, une telle représentation ne permet pas d'exhiber les interactions entre les éléments du réseau étudié. Ainsi, une autre façon de représenter un log est de le faire à l'aide d'un graphe, dont les sommets correspondraient aux alarmes et les arcs seraient définis d'après une relation de causalité, (la notion de corrélation correspondant alors à une version non-orientée des arcs).

De même, [Viard et al., 2018] modélise le trafic IP par un graphe temporel, c'est-à-dire un graphe dans lequel les arcs ont des temps d'activations : les composants du réseau sont les sommets et les arcs représentent des échanges de paquets [Latapy et al., 2018]. L'étude de la structure permet alors de détecter des comportements anormaux au sein du système. Les graphes temporels sont des structures suffisamment génériques pour servir à modéliser un log plus fidèlement qu'avec une simple chaîne de caractères.

[Aghasaryan et al., 1998] modélise un log d'alarmes à l'aide d'un réseau de Petri pseudo-stochastique et acyclique. Pour ça, des « tuiles élémentaires » sont définies : le modèle est construit en associant les tuiles les unes aux autres d'après le log. Cette représentation est utilisée pour l'analyse de causes racines : la trajectoire au sein du modèle la plus vraisemblable est retournée, chaque nouvelle tuile étant ajoutée de sorte à maximiser cette vraisemblance.

1.2 Concepts clés pour le diagnostic de fautes

1.2.1 Quelques mots sur le concept de causalité

Bien que la notion de cause d'un événement semble intuitive, en expliciter une définition est un problème difficile. La causalité a été particulièrement étudiée en philosophie, à une époque où il n'était pas rare d'être à la fois scientifique et philosophe.

Au XVIIe siècle, deux thèses s'opposent sur la nature de la causalité : l'une idéaliste, portée par Isaac Newton et l'autre empiriste portée par George Berkeley, John Locke et David Hume. Selon Newton, le système général du monde est ainsi explicable à partir de principes mathématiques. Dans son *traité de la nature humaine* [Hume, 1739], Hume lie quant à lui la causalité à la notion d'*habitude* : la relation de causalité entre deux événements est issue de l'observation répétée de ces phénomènes. Par exemple, observer cent fois que l'éclair précède le tonnerre, permet d'induire une relation de causalité entre éclairs et tonnerre [1]. Cependant, ce ne sont que des associations qui n'apportent pas de synthèse : cent exemples ne sauraient constituer une preuve, ce qui plonge Hume dans le scepticisme. À la lecture de Hume, Emmanuel Kant sort de son « sommeil dogmatique » et écrit sa *critique de la raison pure* [Kant, 1781]. Il y introduit les *catégories de l'entendement* qui peuvent être vues comme des connecteurs logiques entre deux phénomènes. Par exemple, la causalité est un connecteur. Ces connecteurs sont définis *a priori* mais découverts *a posteriori*. Ainsi, un néophyte peut déduire les règles du jeu d'échecs en observant plusieurs parties sachant que ces règles existent déjà *a priori* de manière abstraite. Le travail du scientifique, qui est de relier des phénomènes à première vue indépendants, est alors constitué d'allers-retours

1. On retrouve ici l'application d'une règle heuristique assez générale : « pour expliquer un phénomène, on préfère les hypothèses suffisantes les plus simples ». Cette règle est appelée *principe de parcimonie* ou *rasoir d'Ockham* et est attribuée au philosophe du XIVe siècle Guillaume d'Ockham. Par exemple, la notion de causalité apporte une description simple du fait que le tonnerre succède habituellement à l'éclair.

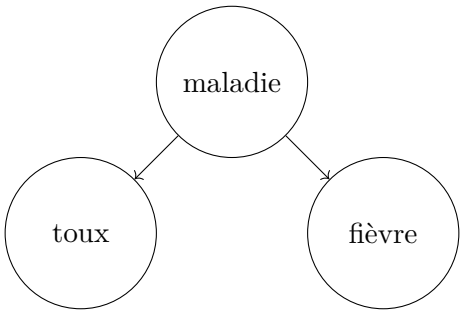


FIGURE 1.1 – Exemple de lien de corrélation qui n'est pas un lien de causalité. Les arcs illustrent ici un lien de causalité : toux et fièvre sont corrélées mais non causales.

entre l'expérience et l'abstraction.

Les propriétés de la causalité en physique sont ainsi très proches de celles définies en philosophie. En effet, de nombreux scientifiques de l'époque étaient aussi philosophes. Ainsi, Leibniz introduit le principe de raison énonçant le fait que *rien n'arrive sans raison* [Leibniz, 1710]. Autrement dit, tout a une cause. Parmi les autres propriétés communément rattachées à la causalité, l'effet précède toujours la cause et ceux-ci sont contigus, c'est-à-dire qu'ils sont toujours proches dans le temps comme dans l'espace.

Bien plus tard, Granger définit la causalité en économétrie [Granger, 1969]. Cette définition est elle-même une héritière des idées des philosophes cités juste avant. Considèrant deux séries temporelles X et Y, X cause Y au sens de Granger si et seulement si connaître le passé de X change la prédiction de Y que l'on aurait fait à partir de son seul passé.

Toutefois dans notre cas, nous cherchons des liens de causalité parmi une multitude d'événements au sein d'une seule série. Pour cela, nous introduirons dans le chapitre 3 une notion relâchée de causalité, qui repose essentiellement sur la coïncidence et la précédence de deux événements.

Enfin, il est important de mentionner que corrélation et causalité ne sont pas équivalentes. Plus exactement, causalité implique corrélation mais pas le contraire. Par exemple, avoir de la fièvre et tousser peuvent être corrélés mais ni la fièvre ne cause la toux ni l'inverse (cf. figure 1.1).

Ainsi, il peut être nécessaire de développer des solutions différentes selon le type de relations entre événements que l'on souhaite établir. Par exemple, on cherche plutôt en

sequence pattern mining à établir des relations causales tandis qu'apprendre des règles d'association parmi un ensemble de transactions consiste à établir un degré de corrélation entre chaque élément. D'un point de vue probabiliste, l'apprentissage de règles d'association peut être vu comme l'étude d'une loi multivariée appelée transaction. Chaque marginale correspond à un objet et une règle d'association définit la corrélation entre plusieurs objets. Des algorithmes tels qu'Apriori [Agrawal et al., 1994], FP-Growth [Han et al., 2000] ou encore Eclat [Zaki, 2000] permettent ainsi de retrouver les ensembles d'objets fréquents afin d'en extraire des règles d'associations.

1.2.2 Interprétabilité : comprendre le résultat d'un algorithme

Les réseaux sont aujourd'hui principalement administrés par des experts humains : le dialogue entre l'homme et la machine doit donc être facilité afin d'assurer le bon fonctionnement du système dans son ensemble. Nous appelons donc *interprétabilité* le fait que le résultat d'un algorithme puisse être compris et expliqué par un agent humain. Dans le cadre de l'analyse de causes racines, l'administrateur doit ainsi pouvoir comprendre quelle est la source d'une panne pour la résoudre efficacement.

Selon Molnar, il n'existe pas de définition canonique et rigoureuse de l'interprétabilité [Molnar, 2020]. Au contraire, cette notion repose sur une intuition que l'on suppose universelle : ainsi une solution est interprétable si elle permet de comprendre « comment » et « pourquoi » une certaine décision a été prise. Une autre façon de concevoir l'interprétabilité est de répondre à la question suivante : est-ce qu'un être humain pourrait prendre une décision similaire ?

Bien que ce concept reste vague, l'interprétabilité reste un critère important dans de nombreux cas d'usages tels que l'administration réseau. On peut également la retrouver dans le domaine des assurances ou dans le domaine médical. Dans ces cas, il est crucial de pouvoir expliquer à des utilisateurs humains non experts la raison d'une décision prise par l'algorithme. Dans le cadre des réseaux de télécommunication, la complexité des systèmes mis en jeu rend la contrainte d'interprétabilité primordiale même lorsqu'elle s'adresse à un utilisateur expert. Cependant, si l'interprétabilité permet de dégager de la simplicité dans un problème complexe, cette simplicité est souvent faite en dépit des performances.

1.3 Analyse de causes racines

Afin d'améliorer la disponibilité d'une architecture au sens large, il est important de pouvoir identifier efficacement les pannes qui surviennent ainsi que leur origine. Il peut s'agir d'un réseau de télécommunication ou d'une architecture logicielle.

La panne d'un composant du réseau peut engendrer la panne d'autres composants voisins : une panne se manifeste donc comme une cascade de fautes qui s'étend dans le temps et dans le réseau. Il est par ailleurs possible que d'autres pannes et erreurs mineures surviennent indépendamment, ce qui en complique encore l'analyse. L'*analyse de causes racines* consiste donc à retrouver la cause première d'une panne parmi une multitude d'erreurs.

Toutefois, du fait de la complexité des réseaux, en taille comme en technologies, retrouver la cause racine d'une panne reste un problème difficile. Cette tâche s'avère laborieuse pour les administrateurs réseaux et les besoins dans des solutions d'analyse de causes racines sont très présents.

1.3.1 Techniques basées sur l'analyse de mots

Regroupement de problèmes similaires

Catégoriser les logs par type de pannes permet d'associer le log décrivant la panne courante à d'autres logs correspondant à des pannes connues et résolues. Bien que la plupart des travaux évoqués dans ce paragraphe soient généralement appliqués à des chaînes de caractères ou en bio-informatique, ils peuvent également s'appliquer à des logs réseau. Étant donné un incident, ils permettent de retrouver dans une base de connaissances des problèmes similaires qui ont été résolus par le passé.

Ainsi, l'algorithme de Smith-Waterman [Smith et al., 1981] permet d'aligner deux mots d'après le problème de recherche de la plus longue sous-suite commune (voir exemple 1). Autrement dit, il s'agit de déterminer un *alignement*, c'est-à-dire la sous-suite d'indices correspondant au plus long sous-mot commun. Il repose sur le calcul d'un score d'alignement (défini à la discrétion de l'utilisateur selon les correspondances, les non-correspondances et les trous) entre les deux mots étudiés : une matrice contenant le score d'alignement entre les préfixes des deux mots est construite récursivement. Ce calcul est réalisé par programma-

tion dynamique. Cette matrice est ensuite parcourue du coin inférieur droit (contenant le score d'alignement des deux mots entiers) vers le coin supérieur gauche (contenant le score d'alignement pour les deux premiers caractères) afin de retrouver la plus longue sous-suite commune. La complexité de cet algorithme est quadratique.

Exemple 1 *La plus longue sous-suite commune aux mots* `xxaxxbcxxxdxx` *et* `ayyybyyyycyyd` *est* `abcd`*.*

Comme l'algorithme de Smith-Waterman, celui de Hunt-Szymanski [Hunt and Szymanski, 1977] permet de retrouver la plus longue sous-suite commune à deux mots donnés, ainsi que sa position. Plutôt que de considérer l'ensemble de tous les alignements possibles, l'algorithme travaille à partir des alignements exacts de caractères qu'il est possible de faire. Cela évite de considérer des cas non pertinents et de passer d'une complexité quadratique à une complexité en $O(n \log(n))$ où n est la taille des mots en entrée.

Dans le domaine de la gestion de pannes, [Cheng et al., 2013] utilise une version modifiée de l'algorithme de Smith-Waterman pour comparer des flux d'alarmes entre eux. Cela permet ainsi de regrouper ces flux par type de panne. Pour cela, l'algorithme de Smith-Waterman a été modifié de façon à intégrer les dates des alarmes afin de relâcher la notion d'ordre d'apparition lorsque deux alarmes ont été émises à peu près simultanément.

[Johannesmeyer et al., 2002] recherche à l'intérieur d'un ensemble de suites de caractères celles similaires à une suite donnée. En pratique, cet ensemble correspond à une base de référence contenant des logs connus. Cette base est confrontée au log contenant la panne courante à identifier. En l'occurrence, les auteurs cherchent à retrouver des motifs anormaux dans l'ensemble des données générées par leur système (un réacteur chimique). Pour cela, la solution découpe des flux d'événements en tronçons qui sont alors comparés à une base de référence. Cette comparaison repose sur des méthodes statistiques centrées autour de l'analyse en composante principale. L'analyse de causes racines est faite par un expert qui peut ainsi rechercher des motifs anormaux dans le système fautif en fonction de motifs anormaux identifiés lors de pannes antérieures. Cela nécessite d'avoir une base de données, suffisamment riche pour que la panne étudiée ne soit pas un événement rare.

L'algorithme BLAST [Altschul et al., 1990] ne cherche pas la sous-suite la plus longue mais la plus « pertinente ». Tout comme Smith-Waterman, une fonction de score d'alignement est donnée en paramètre. Les alignements qualifiés de pertinents sont ceux qui offrent

un score et une vraisemblance élevés. Cet algorithme reçoit en entrée un mot et une requête (sous la forme d'un mot). La requête est décomposée en sous-mots qui sont pondérés selon ce score. Dans un premier temps, l'alignement de séquences n'est réalisé qu'avec les sous-mots les plus pertinents. L'ensemble de ces sous-mots sont déterminés en comparant leur score d'alignement à un seuil donné en paramètre. Ces sous-mots sont étendus avec le reste des sous-mots de la requête : en quantifiant l'espérance mathématique de chacun des alignements obtenus, l'algorithme renvoie finalement les alignements les plus pertinents. La solution retournée par cet algorithme n'est pas optimale contrairement à celle apportée par Smith-Waterman, mais elle s'avère plus rapide.

Recherche de motifs arbitraires

L'analyse d'un log de grande taille est souvent complexe. Retrouver efficacement des motifs spécifiques au sein d'un log permet d'en faciliter l'analyse. Cela nécessite de connaître les motifs à rechercher. Puisqu'un log peut être considéré comme une suite d'alarmes, les outils de la théorie des langages peuvent s'appliquer dans un contexte d'analyse de causes racines.

L'algorithme d'Aho et Corasick [Aho and Corasick, 1975] permet de rechercher des mots dans une chaîne de caractères. Ces mots sont stockés dans un dictionnaire fini. Une structure d'arbre est construite à partir de ce dictionnaire, puis exploitée pour retrouver l'ensemble des mots désirés dans le mot étudié. Cette structure peut être vue comme un automate fini enrichi par des liens suffixes. Un lien suffixe est une application qui lie un sommet, identifiant une chaîne de caractères, à celui identifiant le plus grand suffixe strict de cette chaîne. La structure ainsi construite est parcourue selon le mot donné en entrée. La position des motifs retrouvés est également retournée.

L'algorithme de construction de Thompson [Thompson, 1968] est une généralisation de l'algorithme d'Aho-Corasick et permet de retrouver des motifs définis par une expression rationnelle dans un mot donné en entrée. L'algorithme transforme d'abord l'expression rationnelle en un automate fini non déterministe. Pour cela, il procède récursivement, en associant chaque opération de l'expression rationnelle à un modèle type d'automate. Lorsque cet automate est parcouru selon le mot d'entrée, les motifs retrouvés et leurs positions sont retournés. Il s'agit d'un composant essentiel de `grep`, dont Thompson est également l'auteur. Ces outils ont notamment inspiré le chapitre 4.

Dans le cadre de la détection de comportements anormaux, [Van Lunteren, 2006] a un fonctionnement proche de l'algorithme de Aho-Corasick dans le sens où il repose sur la construction d'un automate fini à partir d'une chaîne de caractère ou d'une expression régulière donnée en paramètre. Cet automate est utilisé pour chercher les motifs correspondant dans un flux de caractères donné en entrée. La définition de cet automate se fait à un niveau matériel ce qui permet de modifier à moindre coût la requête en paramètre et d'accélérer la recherche de motifs par un calcul distribué.

Recherche de motifs fréquents

Dans certains cas, avoir une idée précise des motifs à rechercher est impossible. Bien que les logs soient souvent bruités et difficilement lisibles, les motifs fréquents sont sans doute révélateurs de causalités.

GSP (pour *Generalized Sequential Patterns*) [Srikant and Agrawal, 1996] permet ainsi de rechercher les motifs fréquents parmi un ensemble de suites d'événements. Un événement est ici un ensemble (non ordonné) d'objets atomiques. Ainsi une série d'événements représente une succession de transactions correspondant aux objets de chaque événement. La fréquence d'un motif est définie comme le nombre de suites d'événements contenant ce motif. Un motif est alors qualifié de fréquent par rapport à un seuil donné en paramètre. L'algorithme commence par déterminer l'ensemble des motifs fréquents de taille 1 (les lettres). Puis, récursivement, l'algorithme détermine les motifs fréquents de taille k en fonction de ceux déjà déterminés de taille $k-1$. La récursion se termine dès qu'un k a été atteint pour lequel il n'existe pas de motif fréquent. L'ensemble des motifs fréquents de longueur au plus k est alors retourné.

Comme GSP, SPADE (pour *Sequential PAttern Discovery using Equivalence classes*) [Zaki, 2001] est un outil qui permet de rechercher des motifs fréquents parmi un ensemble de chaînes de causalité. SPADE est une amélioration de GSP car il permet une seule passe sur l'ensemble des chaînes. Ces deux outils sont d'ailleurs inspirés de l'outil de fouille de bases de données Apriori [Agrawal et al., 1994]. Il repose sur la gestion d'un treillis organisé en niveaux : le k-ième niveau contient tous les motifs de longueur k. Chaque sommet correspond à un motif et est pondéré par sa fréquence. Les arcs représentent quant à eux une relation d'inclusion entre les motifs. Ils lient uniquement un sommet vers un sommet du niveau supérieur.

Considérer un log d'alarme comme une simple suite d'événements est une représentation parfois trop simpliste. En effet, dans un même log, plusieurs chaînes de causalités peuvent s'entremêler, du bruit peut être présent. Des outils employant des modèles causaux plus complexes peuvent être considérés. Notamment, une chronique est une structure de données capable de stocker des motifs causaux temporisés [Dousson et al., 1993]. Plus précisément, il s'agit d'une structure de graphe étiqueté sur les sommets et sur les arcs. Les sommets sont étiquetés par des noms d'alarmes, tandis que les arcs sont étiquetés par des intervalles de temps afin de caractériser un lien de causalité entre deux alarmes.

Par exemple, [Dousson and Duong, 1999] recherche les chroniques fréquentes à l'intérieur d'un log. Comme pour GSP, l'algorithme est initialisé à partir des alarmes seules et de leur fréquence. Les motifs de taille k sont déterminés récursivement à partir des motifs fréquents de taille $k-1$ (pour cela, un seuil est donné en paramètre). Bien qu'il n'en soit pas fait mention, le fonctionnement de cet algorithme n'est pas sans rappeler GSP et SPADE.

Dans [Subias et al., 2014], les auteurs recherchent les motifs fréquents d'une collection de logs. Un motif est représenté par une chronique. Pour cela, toutes les interactions observées entre les alarmes sont stockées dans un ensemble d'arbres : un arbre caractérise les interactions entre deux alarmes particulières, ses sommets correspondent à chaque interaction observée et ses arcs à des relations d'inclusion entre les interactions. Ne sont conservées que les interactions qui sont jugées fréquentes pour chaque log. Une fois cette structure de donnée établie, l'algorithme détermine itérativement les chroniques fréquentes à partir des interactions observées.

Stockage de tous les motifs observés

Nous pouvons supposer qu'une panne fréquente est soit anodine, soit qu'elle doit être traitée en priorité. Il peut cependant survenir des pannes plus rares qu'il convient également de traiter. Dans ce cas, considérer seulement les alarmes fréquentes n'est pas suffisant. Une solution pourrait donc être de stocker dans un premier temps l'ensemble des motifs causaux observés. Certains algorithmes issus de la théorie des langages permettent justement de stocker les sous-mots observés dans un mot. Il est par exemple possible de stocker les suffixes d'un mot, caractère par caractère, dans une structure d'arbre appelée *arbre des suffixes*. Un tel arbre contient aussi tous les sous-mots observés dans le mot dont il est issu.

L'algorithme de McCreight [McCreight, 1976] construit ainsi l'arbre des suffixes d'un mot. Par souci de compacité, chaque arc encode un mot. Les suffixes sont ajoutés à l'arbre en commençant par le plus long (le mot entier) et en terminant par le plus court (la dernière lettre). À l'intégration, d'un nouveau suffixe, si une branche partage un même préfixe avec le suffixe en question, alors le suffixe est intégré dans la structure à partir de cette branche. Un système de liens suffixes (un pointeur d'un sommet vers un autre) est maintenu pour en faciliter la localisation des embranchements de la structure ainsi que sa mise à jour. L'exécution de cet algorithme nécessite ainsi de connaître le mot entier, ce qui est pénalisant dans le cas où les caractères doivent être traités à leur réception. Cet algorithme s'exécute en temps linéaire.

Comme McCreight, [Ukkonen, 1995] construit lui aussi l'arbre des suffixes d'un mot donné en entrée. Les arcs encodent également un sous-mot du mot d'entrée : en pratique, il suffit de stocker les indices marquant le début et la fin du facteur correspondant. La complexité en temps est également linéaire. Toutefois, contrairement à McCreight, l'algorithme d'Ukkonen opère en ligne : la structure est construite en découvrant le mot caractère par caractère du premier au dernier : il maintient un ensemble de sommets dits *actifs* et la notion de lien suffixe permet de retrouver efficacement les sommets à partir desquels la structure grandit. Le fait de stocker les indices de début et de fin d'un sous-mot sur un arc permet de ne pas systématiquement mettre à jour les liens suffixes : cela est déjà fait à travers la mise à jour des indices. La figure 1.2 illustre le fonctionnement de l'algorithme d'Ukkonen sur un exemple. En particulier, on peut remarquer qu'il est facile de retrouver tous les états actifs grâce aux liens suffixes et la feuille active la plus profonde. Une fois le mot totalement découvert, les sommets actifs correspondent aux états terminaux de l'automate décrivant le langage des suffixes du mot étudié (les étiquettes ont été déportées des arcs vers les sommets pour plus de commodité). À chaque fois qu'une nouvelle lettre est découverte, la structure est étendue à partir des sommets actifs. Cet algorithme a un rôle important dans nos travaux puisque les algorithmes présentés dans le chapitre 3 en sont des généralisations.

Là encore, la seule structure de mot n'est pas suffisante pour modéliser correctement la notion de causalité dans un log.

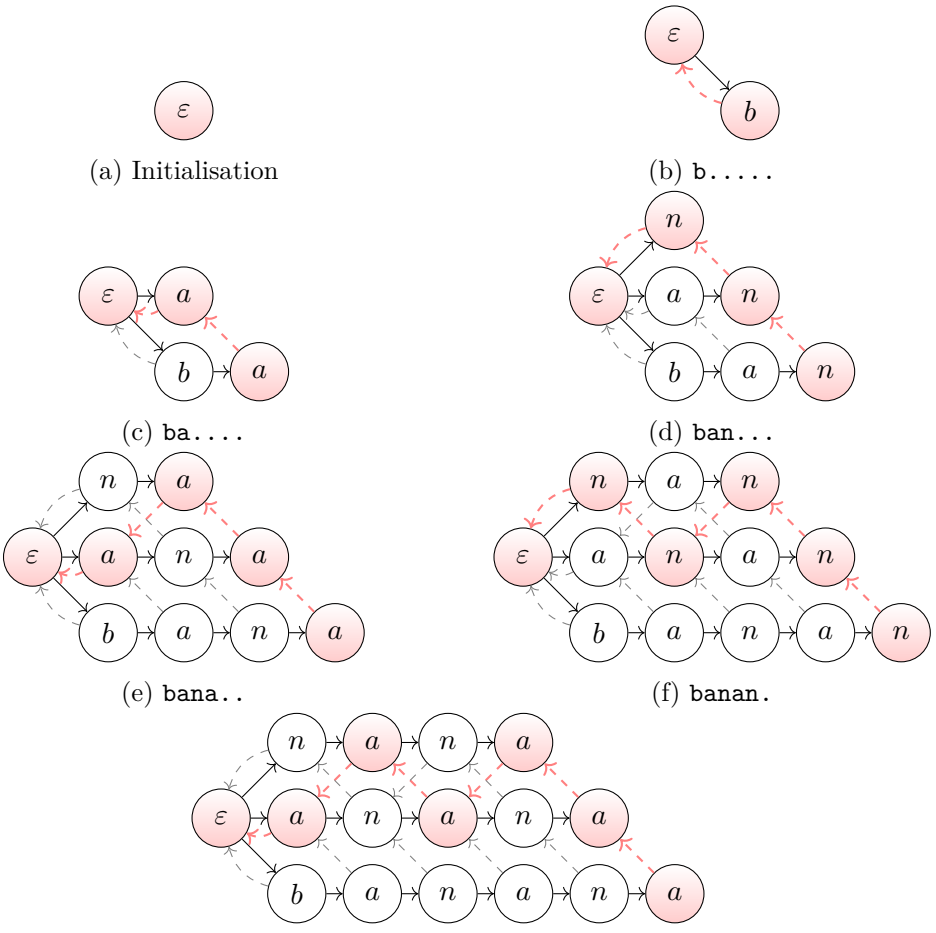


FIGURE 1.2 – Le mot `banana` est découvert lettre par lettre, son arbre des suffixes est construit en ligne grâce à l'algorithme d'Ukkonen. Les arcs en pointillés correspondent aux liens suffixes et les sommets roses correspondent aux états actifs. Ces derniers correspondent aux sommets traversés par le chemin de liens suffixes partant de la feuille du suffixe le plus long. Différemment de la version originale et plus dans l'esprit de ce qui sera proposé dans le chapitre 3, les étiquettes ont ici été déportées sur les sommets. Autre entorse à l'algorithme original, ce sont des lettres qui y sont stockées et non pas des sous-mots.

1.3.2 Méthodes par apprentissage

Recherche de corrélations

Dans cette section, les outils présentés établissent des liens de corrélations entre les signaux d'alarme observés.

Dans [Zhang et al., 2019], les auteurs considèrent plusieurs séries temporelles issues des capteurs d'un même système. Ils en déduisent la matrice de corrélation à chaque instant. L'idée est alors d'entraîner un auto-encodeur convolutionnel basé sur le mécanisme d'attention pour prédire la suite des matrices de corrélation. Cela permet de lier les séries (fautives notamment) entre elles et de détecter les comportements anormaux. Bien qu'il s'agisse d'un modèle non supervisé prenant en compte les dépendances temporelles, les résultats sont difficiles à interpréter.

Afin de réduire la taille des logs étudiés, [Julisch, 2003] regroupe les alarmes d'un log selon leurs attributs, spatiaux comme temporels. Pour chaque attribut, une structure de graphe orienté acyclique est fournie en paramètre afin de hiérarchiser les valeurs observées en un ensemble de propriétés. Chaque sommet correspond à une propriété, et un arc décrit une relation d'inclusion entre ces propriétés. Par exemple, si le jour de la semaine est un attribut, alors `samedi` et `dimanche` seraient des feuilles et auraient pour parent la propriété `weekend`. Tant que l'objectif fixé en termes de taille de log n'est pas rempli, l'algorithme détermine un attribut choisi heuristiquement et fusionne les alarmes selon la hiérarchie de cet attribut.

[Aghasaryan et al., 2018] établit une matrice de corrélation entre les alarmes d'un log. En seuillant cette matrice, une première structure de graphe représentant les liens entre alarmes est obtenue. Cette structure peut alors être utilisée pour la construction d'un modèle causal à l'aide d'outils tels que ceux présentés dans [Aghasaryan et al., 1998] et détaillés en section 1.3.1.

Recherche de causes racines

Finalement, les outils présentés dans cette section s'attaquent directement au problème de recherche de causes racines. Les logs fournissant des données abondantes, l'apprentissage

automatique semble être une piste toute indiquée pour la gestion des réseaux. De nombreux travaux ont d'ailleurs été proposés dans ce sens comme il est reflété à travers les études [Solé et al., 2017] et [Wang et al., 2020].

[Smith et al., 2008] caractérise les intrusions faites sur un système. L'analyse a lieu en deux temps : dans un premier temps, les alertes provenant du système sont regroupées une première fois de sorte à caractériser une étape d'attaque, ces étapes sont dans un second temps elles-mêmes regroupées pour caractériser une attaque complète. Les outils retenus pour chacune des deux étapes sont respectivement un auto-encodeur et l'algorithme *Expectation-Maximization* [Dempster et al., 1977, Dellaert, 2002].

[Weidl et al., 2005] propose une méthodologie d'analyse de cause racine à partir de la construction d'un réseau bayésien dans un style orienté-objet. La définition de fonctions « bayésiennes » aide l'administrateur du système à concevoir un modèle bayésien pur le diagnostic de faute à partir de ses connaissances *a priori*. Le réseau résultant est ensuite entraîné à partir de logs émis par le système, et permet d'assister l'expert lors de l'identification et de la correction d'une panne.

[Sorsa and Koivo, 1993] enquête sur la performance de plusieurs types de réseaux de neurones dans le cadre du diagnostic de fautes d'un simulateur d'échangeur thermique. En particulier, les auteurs ont entraîné un perceptron multicouches à faire relier les symptômes observés à leurs causes. Le caractère supervisé de la tâche est néanmoins une limitation forte. En effet, les réseaux connaissent une grande diversité de pannes et construire une base de données exhaustive munie d'une vérité terrain est impossible en pratique. C'est pourquoi les auteurs ont également entraîné des modèles non supervisés tels que des cartes auto-organisantes (SOM) pour regrouper les symptômes observés entre eux. L'espace d'observation est projeté sur une surface en deux dimensions et les clusters sont établis *a posteriori*. Cependant, il est difficile d'attacher du sens à ces clusters.

Les réseaux bayésiens sont aussi communément employés en analyse de causes racines. Par exemple, dans [Alaeddini and Dogan, 2011], les auteurs séparent causes et symptômes dans un réseau bayésien biparti. Les symptômes sont décrits soit par certaines caractéristiques soit par certaines règles. Bien que le cadre probabiliste favorise l'interprétabilité des résultats, les réseaux bayésiens font généralement face à des problèmes de passage à l'échelle. En effet, accroître la complexité du système augmente fortement la quantité de mémoire nécessaire au stockage des probabilités conditionnelles.

Afin d'expliquer des requêtes fautives au sein du système de service Internet eBay, [Chen et al., 2004] entraîne un arbre de décision à classifier les fautes et les succès. Une fois entraîné, le chemin correspondant à une requête fautive est alors utilisé comme description de sa cause racine. En général, la simplicité du modèle fait que les arbres de décisions sont interprétables. Toutefois, accroître la complexité du système étudié implique souvent une instabilité de la phase d'entraînement [Aluja-Banet and Nafria, 2003], ce qui en affecte l'interprétabilité.

Inférence d'un modèle sous-jacent

La connaissance de la topologie du réseau n'étant pas toujours disponible, certains travaux cherchent à retrouver cette information à partir d'un log d'alarmes.

Ainsi, étant donné un graphe dont la topologie est inconnue, [Gomez-Rodriguez et al., 2012] considère un log où chaque ligne décrit la contamination d'un sommet à une certaine date. L'objectif est alors de retrouver la structure de graphe la plus vraisemblable. Pour cela, les auteurs définissent un modèle génératif épidémiologique.

Interpréter le résultat d'un algorithme « boîte noire »

L'étude [Wang et al., 2020] mentionne de nombreuses utilisations de réseaux de neurones appliqués à des données spatio-temporelles, mais remarque le manque d'interprétabilité de ces solutions, souvent considérées comme des « boîtes noires ». Ainsi, des outils permettant d'interpréter *a posteriori* des résultats *a priori* non interprétables viennent au secours des solutions mentionnées précédemment. Principalement, ces outils explicitent le lien entre les entrées, localement à la décision prise par le système « boîte noire » étudié.

Ainsi, LIME (pour *Local Interpretable Model-agnostic Explanations*) [Ribeiro et al., 2016] permet d'expliquer localement une classification. Étant donné un classificateur et l'entrée classifiée, l'outil entraîne un séparateur linéaire avec pour base d'entraînement des entrées perturbées autour de l'entrée originale et pour étiquettes les réponses correspondantes du classificateur « boîte noire ». Ce séparateur linéaire est alors utilisé comme explication puisqu'il permet de lier les entrées par des règles très simples, bien que grossières.

Étant donné un modèle, SHAP (pour *Shapley Additive exPlanations*) [Lundberg and Lee, 2017] confère à chaque caractéristique d'entrée du modèle un score d'importance pour une

décision donnée. Le modèle est réentraîné sur un sous-ensemble de caractéristiques d'entrée : si la décision correspondante est différente de la décision faite en prenant compte de toutes les caractéristiques, alors les caractéristiques retirées avaient une importance dans la prise de décision du modèle.

En l'état, ces outils semblent néanmoins peu appropriés aux données temporelles et encore moins à des structures causales. Par ailleurs, à chaque intervention de l'expert, ces outils doivent être mis en place en plus de l'utilisation de l'algorithme.

Chapitre 2

Modèles pour la prédiction de données temporelles

Étant donnés un ou plusieurs logs contenant un ensemble d'alarmes observées sur un réseau, nous cherchons à prédire quelles seront les prochaines alarmes émises par le système. Or, ces logs sont des fichiers qui dressent l'historique des événements au sein du système. Sous ce point de vue, le problème de prédiction dans un log se formalise alors comme un problème classique de prédiction de données temporelles. Pour commencer, étudions le problème de prédiction dans le cas statique. Considérons la réalisation de variables aléatoires (x, y) à partir d'une loi jointe $p(x, y)$ [1] : à partir de l'observation de x, nous souhaitons prédire y. Nous cherchons ainsi un estimateur $\hat{y} = f(x)$ tel que $\hat{y}$ soit « proche » de y en un sens qu'il reste à préciser.

2.1 Introduction à la prédiction de séries temporelles

2.1.1 Prédiction dans le cas d'une loi jointe connue

Supposons dans un premier temps la loi jointe $p(x, y)$ connue. Dans un contexte bayésien, la construction de l'estimateur $f(.)$ est déterminée par le choix d'une *fonction de coût* $L(.,.)$

1. Tant que cela ne posera pas d'ambiguité, nous confondrons variables aléatoires et réalisations. De même, selon le contexte, $p(.)$ peut désigner une probabilité ou une densité.

dont le but est de quantifier l'erreur entre la prédiction $f(x) = \hat{y}$ et la variable cachée y. Le choix de cette fonction de coût dépend de la nature du problème de prédiction. Par exemple, si l'on cherche à prédire la position d'un objet dans l'espace, une fonction de coût basée sur une distance euclidienne semble naturel. Une fois cette fonction de coût choisie, le risque bayésien est défini comme l'espérance du coût entre $f(x) = \hat{y}$ et y :

$$\begin{aligned} R(f) &= \mathbb{E}\left[L(f(x), y)\right] \\ &= \int_{x,y} L(f(x), y) p(x, y) \mathrm{d}x \mathrm{d}y. \end{aligned} \tag{2.1}$$

Ainsi, le problème de construction d'un estimateur devient celui de la minimisation du risque bayésien

$$f^\star = \underset{f}{\operatorname{argmin}}\ R(f). \tag{2.2}$$

On peut montrer que l'estimateur $f^\star$ peut être construit à partir de la loi *a posteriori* $p(y|x) = \frac{p(x,y)}{p(x)}$ (également appelée *loi prédictive* dans le cadre de la prédiction). En effet, l'équation (2.2) se réécrit en

$$f^\star = \underset{f}{\operatorname{argmin}} \int_x p(x) \left(\int_y L(f(x), y) p(y|x) \mathrm{d}y \right) \mathrm{d}x. \tag{2.3}$$

Minimiser (2.2) revient à minimiser $\int_y L(f(x), y) p(y|x) \mathrm{d}y$ par rapport à $f(x)$ à x fixé. La minimisation d'une intégrale double est alors devenue celle d'une intégrale simple. En particulier, pour certaines fonctions de coût classiques $L(.,.)$ (erreur quadratique, erreur en valeur absolue, erreur « tout ou rien », etc.), l'estimateur obtenu $f^\star$ correspond à une caractéristique (moyenne, médiane, mode) de $p(y|x)$.

Exemple 2 *Nous cherchons dans cet exemple à prédire la position y d'un objet après en avoir observé une position x (variables réelles unidimensionnelles). Pour cela, nous choisissons la fonction de coût $L : (y, y') \mapsto (y - y')^2$ qui correspond simplement à la distance euclidienne entre deux positions. Le meilleur estimateur correspondant à cette fonction de coût est l'estimateur qui minimise la quantité*

$$\mathbb{E}\left[L(f(x), y)\right] = \mathbb{E}\left[(\hat{y} - y)^2\right] = \int_x p(x) \left(\int_y (\hat{y} - y)^2 p(y|x) \mathrm{d}y \right) \mathrm{d}x.$$

Minimiser le risque bayésien ci-dessus revient, pour tout x fixé, à minimiser par rapport à

$\hat{y}$ la fonction

$$\int_y (\hat{y} - y)^2 p(y|x)\, dy = \ \mathrm{Var}(y|x) + (\hat{y} - \mathbb{E}[y|x])^2.$$

Cette fonction est minimale lorsque $\hat{y} = \mathbb{E}[y|x]$. Autrement dit, le meilleur estimateur selon le coût quadratique est la moyenne de la loi prédictive $p(y|x)$.

2.1.2 Prédiction lorsque la loi jointe n'est pas connue

En pratique, une difficulté réside dans le fait que la loi jointe n'est en réalité pas connue. Pour résoudre ce problème, il est possible d'approcher le risque bayésien (2.1) de deux façons : soit en cherchant à modéliser la loi jointe $p(x, y)$ par une loi paramétrique $p_\theta(x, y)$, soit en approchant l'intégrale (2.1) à partir de réalisations.

La première approche consiste donc à construire un modèle de la loi p. Pour cela, on se restreint à une famille paramétrée de lois $(p_\theta)_{\theta \in \Theta}$, où le paramètre θ peut être multidimensionnel (par exemple, il peut décrire la moyenne et la variance d'une loi normale). À partir d'un jeu fini de données indépendantes

$$\mathcal{E} = \left\{(x_i, y_i) \overset{\text{i.i.d.}}{\sim} p(x, y)\right\}_{i \in [\![1,n]\!]} \tag{2.4}$$

issues de la loi inconnue $p(x, y)$, la qualité de p_θ peut être quantifiée par la fonction de *vraisemblance* [Borovkov, 1987, Wasserman, 2013] :

$$\mathcal{L}(.; \mathcal{E}) : \theta \mapsto \prod_{i=1}^{n} p_\theta(x_i, y_i). \tag{2.5}$$

La fonction de vraisemblance répond intuitivement à la question suivante : est-ce que la loi p_θ aurait pu raisonnablement générer l'ensemble des paires (x_i, y_i) ? L'approximation de la loi jointe se résume alors à déterminer le paramètre θ qui maximise la vraisemblance. Toutefois, le choix de la famille paramétrique est crucial : il faut que p_θ puisse correctement modéliser les données récoltées tout en permettant le calcul des quantités clés pour l'apprentissage et l'estimation.

Deux problèmes sont associés à la fonction de vraisemblance : son calcul (voir équation (2.5)) et sa maximisation. Notamment, pouvoir calculer la vraisemblance n'implique pas né-

cessairement que l'on puisse la maximiser. En général, la maximisation de la vraisemblance n'est pas faisable directement et cela doit être fait de manière approchée. Par exemple, dans le cas de modèles probabilistes à variables latentes (c'est-à-dire des variables non observées visant à enrichir le modèle $p_\theta(x, y)$), la maximiser requiert l'utilisation d'approximations telles que l'algorithme *Expectation Maximization* (voir section 2.2.2).

La seconde approche ne fait pas d'hypothèses sur la loi jointe $p(x, y)$. À la place, on commence par déduire l'estimateur empirique du critère (2.1) à partir du même jeu d'entraînement $\mathcal{E}$ (voir équation (2.4)) [Bishop, 2006, Hastie et al., 2009]. Le problème de construction d'un estimateur devient celui de la minimisation du risque empirique :

$$f_n^\star = \operatorname*{argmin}_f \frac{1}{n} \sum_{i=1}^{n} L(f(x_i), y_i). \tag{2.6}$$

Cependant, puisque le jeu d'entraînement est fini, si aucune contrainte n'est posée alors n'importe quelle fonction interpolant les points (x_i, y_i) répond au problème d'optimisation (2.6). Dans un tel cas, le modèle a surappris et est incapable de généraliser à de nouvelles observations. Pour pallier ce problème, on choisit une famille de fonctions $(f_\theta)_{\theta\in\Theta}$. Le problème posé par (2.6) se réécrit enfin en un problème d'estimation de paramètre :

$$\theta_n^\star = \operatorname*{argmin}_\theta \frac{1}{n} \sum_{i=1}^{n} L(f_\theta(x_i), y_i), \tag{2.7}$$

et on choisit l'estimateur $f(x) = f_{\theta_n^\star}(x)$ (la notation $\theta_n^\star$ reflète le fait que l'estimateur dépend du jeu d'entraînement $\mathcal{E}$ (de taille n). Comme précédemment, le choix de la famille résulte d'un compromis : une famille trop simple ne permettra pas d'obtenir des prédictions pertinentes et réalistes tandis qu'une famille trop complexe et trop flexible présente un risque de surapprentissage. Enfin, le choix de la famille doit permettre l'apprentissage, c'est-à-dire la résolution de l'équation (2.7). Parmi les familles de fonctions classiquement utilisées, on trouve les fonctions appartenant à un espace de Hilbert à noyau reproduisant [Manton and Amblard, 2015, Paulsen and Raghupathi, 2016] et les fonctions définies par un réseau de neurones [Jain et al., 1996, Marilly et al., 2002, LeCun et al., 2015]. Le problème d'optimisation (2.7) pour ces familles de fonctions donne des algorithmes d'apprentissage bien connus tels que la méthode des moindres carrés (linéaire ou par noyau) [Bishop, 2006], les *Support Vector Machines* (SVM) [Burges, 1998, Hu et al., 2003, Vapnik, 2013] ou les algorithmes d'apprentissage profond, [Bishop, 2006, Goodfellow et al., 2016] pour la régression ou la classification.

2.1.3 Prédiction de séries temporelles

Dans un contexte de prédiction de données temporelles, notre objectif est de prédire l'observation à venir x_{t+1} connaissant l'historique des observations $x_{0:t}$, et ce pour tout t. L'application de la méthodologie précédente dans laquelle x devient $x_{0:t}$ et y devient x_{t+1} à tout instant nous amène à considérer $p(x_{t+1}|x_{0:t}) = \frac{p(x_{0:t+1})}{p(x_{0:t})}$ inaccessible (qu'on ne connait pas ou qu'on ne sait pas calculer) quel que soit t. Comme précédemment, nous allons chercher à modéliser la loi $p(x_{0:t})$ et/ou la fonction $f(x_{0:t})$, en prenant en compte le fait que cette modélisation doit donner accès à $p(x_{0:t})$ et/ou $f(x_{0:t})$ pour tout t et pour toute suite $x_{0:t}$. En d'autres termes, la loi ou la fonction ne doit pas se contenter de modéliser les données d'apprentissage.

Considérons maintenant l'adaptation des deux méthodes du paragraphe précédent (estimation de loi ou de fonction) à partir d'un jeu d'apprentissage constitué d'une suite d'observations de longueur T

$$\mathcal{E} = \{x_{0:T} \sim p(x_{0:T})\}\,. \tag{2.8}$$

Alors :

- étant donné un modèle paramétré par θ tel que $p_\theta(x_{0:t})$ est connue pour tout t, la vraisemblance s'écrit

$$\begin{aligned}\mathcal{L}(\theta;\mathcal{E}) &= p_\theta(x_{0:T}) \\ &= p_\theta(x_0)\prod_{t=1}^{T} p_\theta(x_t|x_{0:t-1}),\end{aligned} \tag{2.9}$$

  c'est-à-dire comme le produit des vraisemblances prédictives pour chaque instant ;

- étant donnée une famille de fonctions paramétrées par θ, telle que $f_\theta(x_{0:t})$ est connue quel que soit t, on minimise un risque empirique construit par moyennage sur le temps :

$$\theta_T^\star = \underset{\theta}{\operatorname{argmin}}\ \frac{1}{T}\sum_{t=0}^{T-1} L(f_\theta(x_{0:t}), x_{t+1}), \tag{2.10}$$

  et on choisit l'estimateur $f(x_{0:t}) = f_{\theta_T^\star}(x_{0:t})$.

Finalement, qu'il s'agisse de modéliser la loi sous-jacente des observations ou de déterminer une classe de fonctions à partir de laquelle minimiser le risque empirique, les modèles retenus

doivent eux-mêmes pouvoir exprimer $p_\theta(x_{0:t})$ (respectivement $f_\theta(x_{0:t})$), quelle que soit la suite d'observations $x_{0:t}$ et quel que soit t, à partir d'un paramètre θ qui ne dépend pas du temps. Nous verrons que cette contrainte est en particulier vérifiée par les modèles récursifs que nous introduirons plus loin dans les sections 2.2 et 2.3.2.

Pour conclure, le problème de la modélisation (de f comme de p) répond à un compromis entre simplicité des calculs et réalisme du modèle. Si un modèle simpliste ne donne qu'une vision naïve et peu précise du système étudié, un modèle trop complexe est susceptible de surrapprendre et/ou de rendre les calculs liés à l'utilisation du modèle impossibles en pratique. Notamment, bien que l'exploitation d'un modèle à paramètres connus ne pose pas de difficultés particulières (théoriques du moins), le calcul de la vraisemblance ainsi que l'apprentissage de ses paramètres peuvent s'avérer difficiles. Rappelons enfin que dans le cadre de la prédiction de données, les modèles mis en jeu doivent pouvoir gérer des historiques $x_{0:t}$ croissants et donc décrire le problème de manière récursive.

Dans ce chapitre, nous considérons donc deux catégories de modèles particulièrement bien adaptés à la prédiction de séries temporelles : les modèles markoviens d'une part et les architectures neuronales récurrentes d'autre part. Notamment, toutes deux répondent aux trois critères mentionnés dans le paragraphe précédent : ces modèles sont raisonnablement réalistes, ils permettent l'apprentissage de leurs paramètres à partir de données d'entraînement et finalement, ils permettent un traitement récursif des observations. Les premiers sont des modèles probabilistes visant à approcher la loi jointe $p(x_{0:t})$ inconnue tandis que les seconds composent une classe de fonctions paramétrées généralement utilisées dans le cadre de l'apprentissage automatique explicité par l'équation (2.10). Bien qu'à première vue ces deux modèles ne résultent pas de la même approche et ont été étudiés dans des communautés différentes, nous verrons quels rapprochements peuvent être faits et en quoi leur comparaison est pertinente.

2.2 Modèles markoviens

Le modèle probabiliste le plus simple que l'on puisse envisager est un modèle supposant que toutes les observations sont indépendantes. Cependant, il s'agit d'une modélisation simpliste et non pertinente pour le problème de prédiction de séries temporelles. Tout l'enjeu est en effet de caractériser les dépendances entre observations par un modèle qui soit à la fois suffisamment réaliste et utilisable en pratique.

2.2.1 Chaînes de Markov

Une chaîne de Markov est un modèle probabiliste simple qui repose principalement sur l'*hypothèse de Markov* selon laquelle, conditionnellement à toutes les observations passées $x_{0:t}$, l'observation future x_{t+1} ne dépend que de l'état courant x_t :

$$p_\theta(x_{t+1}|x_{0:t}) = p_\theta(x_{t+1}|x_t). \tag{2.11}$$

Alors, la loi jointe du modèle peut se factoriser en

$$p_\theta(x_{0:t}) = p_\theta(x_0) \prod_{s=0}^{t-1} p_\theta(x_{s+1}|x_s). \tag{2.12}$$

Le jeu de paramètres θ comprend alors les caractéristiques de la loi initiale $p_\theta(x_0)$ et des lois de probabilité de transition $p_\theta(x_{t+1}|x_t)$. Puisque toutes les variables du modèle sont observées (contrairement aux chaînes de Markov cachées présentées dans la section 2.2.2), les paramètres du modèle peuvent directement être estimés : dans le cas discret, il s'agit de calculer les probabilités de transition de manière empirique, tandis que dans le cas continu, ceci est fait en dérivant la vraisemblance par rapport à θ.

2.2.2 Chaînes de Markov cachées

Modèle

Une chaîne de Markov reste cependant un modèle encore trop simple et peu expressif. Autrement dit, les lois caractérisées par un tel modèle ne forment pas une famille assez riche pour espérer représenter convenablement la loi inconnue $p(x_{0:t})$. Afin d'étendre la classe des lois $p_\theta(x_{0:t})$ représentée par une chaîne de Markov, on introduit un processus *latent* $h_{0:t}$ [2]. Chaque h_s peut-être discret ou continu et n'est jamais observé en pratique.

Ces variables latentes peuvent être interprétées de plusieurs manières. D'une part, elles peuvent faire office d'intermédiaires de calcul et ainsi contribuer à complexifier la loi $p_\theta(x_{0:t})$, donc en augmenter le pouvoir modélisant. En effet, le modèle décrit la loi $p(h_{0:t}, x_{0:t})$ dont

2. Nous distinguons ici la notion de variable cachée et celle de variable latente : la première notion fait opposition à celle de variable observée (par exemple, nous n'observons pas ce que nous souhaitons prédire) tandis que la seconde caractérise des variables intermédiaires ajoutées à un modèle dans le but de le complexifier. En pratique, les variables latentes sont cachées mais ce n'est pas nécessairement réciproque.

$p(x_{0:t})$ est une loi marginale. D'autre part, l'ajout de variables latentes peut avoir un intérêt sémantique : avec un modèle approprié, il est possible d'interpréter les états latents. Par exemple, les alarmes observées dans un log sont généralement symptomatiques de causes racines plus profondes. L'ajout de variables latentes pourrait ainsi permettre l'intégration de ces causes racines au modèle.

Il est cependant important pour un modèle de garder une complexité calculatoire raisonnable pour être utilisé en pratique. À cet effet, une chaîne de Markov cachée (*Hidden Markov Model* ou HMM) généralise une chaîne de Markov par l'ajout d'un processus latent de manière parcimonieuse. Un HMM est caractérisé par trois propriétés de dépendances et d'indépendances conditionnelles. Tout d'abord, la suite des états latents d'un HMM est une chaîne de Markov :

$$p_\theta(h_{t+1}|h_{0:t}) = p_\theta(h_{t+1}|h_t). \tag{2.13}$$

Par ailleurs, conditionnellement à tous les états latents, les observations sont indépendantes :

$$p_\theta(x_{0:t}|h_{0:t}) = \prod_{s=0}^{t} p_\theta(x_s|h_{0:t}). \tag{2.14}$$

Enfin, conditionnellement à tous les états latents, les observations ne dépendent que de l'état latent courant :

$$\forall s \in [\![0, t]\!], p_\theta(x_s|h_{0:t}) = p_\theta(x_s|h_s). \tag{2.15}$$

Ces propriétés permettent finalement d'écrire la loi jointe $p_\theta(x_{0:t}, h_{0:t})$ sous forme factorisée :

$$p_\theta(h_{0:t}, x_{0:t}) = p_\theta(h_0) \prod_{s=0}^{t-1} p_\theta(h_{s+1}|h_s) \prod_{s=0}^{t} p_\theta(x_s|h_s). \tag{2.16}$$

Cette factorisation de la loi jointe peut notamment être représentée par le réseau bayésien de la figure 2.1.

Calcul de la vraisemblance

Nous avons pu voir en introduction de ce chapitre que le choix d'un modèle dépendait entre autres de la facilité à calculer la vraisemblance $p_\theta(x_{0:t})$ à paramètre θ donné d'une

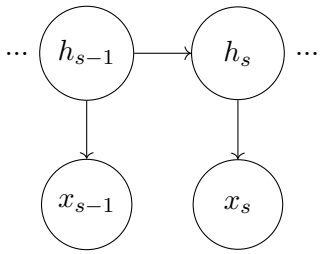


FIGURE 2.1 – Représentation d'un HMM par un réseau bayésien. Pour rappel, un réseau bayésien représente sous la forme d'un graphe dirigé acyclique une loi de probabilité jointe : chaque sommet du graphe correspond à une variable aléatoire x et les arcs sont disposés de telle sorte que la loi jointe se factorise sous la forme $\prod_x p(x|\text{parents}(x))$.

part, et maximiser cette dernière par rapport à θ d'autre part. Rappelons que (d'après le modèle (2.9)), nous savons calculer la vraisemblance à partir des vraisemblances prédictives. Notamment, pour tout $s \in [\![1, t]\!]$ et en utilisant les propriétés du modèle (2.16) :

$$\begin{aligned} p_\theta(x_s|x_{0:s-1}) &= \int_{h_s} p_\theta(h_s, x_s|x_{0:s-1})\mathrm{d}h_s \\ &= \int_{h_s} p_\theta(h_s|x_{0:s-1})p_\theta(x_s|h_s)\mathrm{d}h_s \\ &= \iint_{h_s,h_{s-1}} \underbrace{p_\theta(h_{s-1}|x_{0:s-1})}_{\text{loi de filtrage}} \underbrace{p_\theta(h_s|h_{s-1})p_\theta(x_s|h_s)}_{\text{transitions d'un HMM}} \mathrm{d}h_s\mathrm{d}h_{s-1}, \end{aligned} \tag{2.17}$$

où les facteurs $p_\theta(x_s|h_s)$ et $p_\theta(h_s|h_{s-1})$ correspondent aux transitions d'un HMM (voir équation (2.16)). Le facteur $p_\theta(h_{s-1}|x_{0:s-1})$ correspond quant à lui à la *loi de filtrage* qui se calcule de manière récursive :

$$\begin{aligned} p_\theta(h_s|x_{0:s}) &= \int_{h_{0:s-1}} p_\theta(h_{0:s}|x_{0:s})\mathrm{d}h_{0:s-1} \\ &= \int_{h_{0:s-1}} \frac{p_\theta(h_{0:s}, x_{0:s})}{p_\theta(x_{0:s})}\mathrm{d}h_{0:s-1} \\ &= \int_{h_{0:s-1}} \frac{p_\theta(h_s, x_s|x_{0:s-1}, h_{0:s-1})}{p_\theta(x_s|x_{0:s-1})} p_\theta(h_{0:s-1}|x_{0:s-1})\mathrm{d}h_{0:s-1} \\ &= \int_{h_{0:s-1}} \frac{p_\theta(h_s|h_{0:s-1}, x_{0:s-1})p_\theta(x_s|h_{0:s}, x_{0:s-1})}{p_\theta(x_s|x_{0:s-1})} p_\theta(h_{0:s-1}|x_{0:s-1})\mathrm{d}h_{0:s-1}. \end{aligned} \tag{2.18}$$

Par ailleurs, la structure de chaîne de Markov cachée (voir équation (2.16)) implique également que

$$p_\theta(h_s|x_{0:s-1}, h_{0:s-1}) = p_\theta(h_s|h_{s-1}), \tag{2.19}$$

et

$$p_\theta(x_s|x_{0:s-1}, h_{0:s}) = p_\theta(x_s|h_s). \tag{2.20}$$

Donc l'équation (2.18) se réécrit

$$\begin{aligned} p_\theta(h_s|x_{0:s}) &= \int_{h_{0:s-1}} \frac{p_\theta(h_s|h_{s-1})p_\theta(x_s|h_s)}{p_\theta(x_s|x_{0:s-1})} p_\theta(h_{0:s-1}|x_{0:s-1})\mathrm{d}h_{0:s-1} \\ &= \frac{p_\theta(x_s|h_s)}{p_\theta(x_s|x_{0:s-1})} \int_{h_{s-1}} p_\theta(h_s|h_{s-1})p_\theta(h_{s-1}|x_{0:s-1})\mathrm{d}h_{s-1}. \end{aligned} \tag{2.21}$$

Ainsi les équations (2.17) et (2.21) donnent une façon récursive de calculer la loi prédictive $p_\theta(x_s|x_{0:s-1})$ et la loi de filtrage $p_\theta(h_s|x_{0:s})$. En effet, étant donné la loi $p_\theta(h_0)$, on calcule d'abord $p_\theta(x_s|x_{0:s-1})$ à partir de $p_\theta(h_{s-1}|x_{0:s-1})$ selon l'équation (2.17). On calcule ensuite $p_\theta(h_s|x_{0:s})$ à partir de la loi de filtrage à l'instant précédent $p_\theta(h_{s-1}|x_{0:s-1})$, de la loi prédictive $p_\theta(x_s|x_{0:s-1})$ et des transitions du modèle de Markov caché (2.16) selon l'équation (2.21). En particulier, c'est la structure des chaînes de Markov cachées qui permet (du moins, théoriquement) ce calcul de la vraisemblance ; notons qu'en pratique les intégrales mises en jeu dans les équations (2.17) et (2.21) peuvent s'avérer difficilement calculables.

Apprentissage

Quand bien même la vraisemblance serait calculable en pratique, la maximiser peut s'avérer difficile. En particulier, dans le cas d'un modèle à variables latentes comme c'est ici le cas, le calcul direct de l'estimateur du maximum de vraisemblance est compliqué. On lui préfère généralement une méthode itérative. Ainsi, l'algorithme *Expectation Maximization* (EM) est une méthode approchée pour l'apprentissage des paramètres d'un modèle à variables latentes [Dempster et al., 1977, Dellaert, 2002]. Cet algorithme calcule itérativement une suite de paramètres θ_i. À chaque itération, on calcule d'abord l'espérance de la

log-vraisemblance conditionnellement aux observations $x_{0:t}$ sous un paramètre θ_i :

$$\mathbb{E}_{\theta_i}\left[\log p_\theta(x_{0:t}, h_{0:t})|x_{0:t}\right] = \int_{h_{0:t}} p_{\theta_i}(h_{0:t}|x_{0:t}) \log p_\theta(x_{0:t}, h_{0:t}) \mathrm{d}h_{0:t} \tag{2.22}$$

$$= \sum_{s=1}^{t} \int_{h_{s-1},h_s} p_{\theta_i}(h_{s-1}, h_s|x_{0:t}) \left[\log p_\theta(x_s|h_s) + \log p_\theta(h_s|h_{s-1})\right] \mathrm{d}h_{s-1}\mathrm{d}h_s$$

$$+ \int_{h_0} p_{\theta_i}(h_0|x_{0:t}) \log\left(p_\theta(x_0|h_0)p_\theta(h_0)\right) \mathrm{d}h_0, \tag{2.23}$$

puis on met à jour ce paramètre en θ_{i+1} (autrement dit, on passe de θ_i à θ_{i+1}) en maximisant la quantité

$$\theta_{i+1} = \underset{\theta}{\operatorname{argmax}}\ \mathbb{E}_{\theta_i}\left[\log p_\theta(h_{0:t}, x_{0:t})|x_{0:t}\right]. \tag{2.24}$$

Les équations (2.22) et (2.24) correspondent respectivement aux étapes E et M de l'algorithme EM. En particulier, pour que l'étape E soit réalisable, il est nécessaire que les facteurs $p_{\theta_i}(h_{s-1}, h_s|x_{0:t})$ et $p_{\theta_i}(h_0|x_{0:t})$ puissent être calculés. Cet algorithme garantit la croissance de la vraisemblance au fur et à mesure des itérations :

$$\forall i, p_{\theta_i}(x_{0:t}) \leq p_{\theta_{i+1}}(x_{0:t}). \tag{2.25}$$

Des garanties théoriques plus fortes sont soumises à conditions [Wu, 1983]. En particulier, il n'existe pas de garantie de convergence vers un optimum global.

En pratique, le calcul de la vraisemblance par la loi de filtrage et sa maximisation par l'algorithme EM dépendent des hypothèses du modèle. Nous pouvons distinguer trois cas.

Cas 1 : états continus (système d'état linéaire et gaussien)

Supposons que les suites $x_{0:t}$ et $h_{0:t}$ prennent des valeurs continues et que les transitions $p_\theta(h_{s+1}|h_s)$ et $p_\theta(x_s|h_s)$ sont linéaires gaussiennes :

$$h_{s+1} = F_s h_s + u_s, \tag{2.26}$$

$$x_s = G_s h_s + v_s, \tag{2.27}$$

où F_s et G_s sont des matrices et (u_s, v_s) est un vecteur aléatoire gaussien

$$\begin{bmatrix} u_s \\ v_s \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} Q_s & 0 \\ 0 & R_s \end{bmatrix}\right). \tag{2.28}$$

La variable latente h_0 suit également une loi normale. On retrouve souvent ces hypothèses dans des applications telles que la navigation [Harvey, 1990]. Le cas linéaire gaussien est un cas favorable car, sous ces hypothèses, toutes les lois sont gaussiennes et donc la propagation récursive des lois à travers le temps se résume à la propagation des paramètres de ces dernières (en l'occurrence leurs vecteurs moyennes et matrices de covariance).

Le calcul de la vraisemblance dans le modèle décrit par les équations (2.26),(2.27) et (2.28) peut se faire par l'intermédiaire du filtre de Kalman. Il s'agit d'un outil apparu dans les années 1960 en automatique [Kalman, 1960, Kalman and Bucy, 1961, Ho and Lee, 1964], qui a par la suite été abondamment étudié [Anderson and Moore, 2012, Kailath et al., 2000, Meinhold and Singpurwalla, 1983].

Le filtre de Kalman permet le calcul récursif de la loi de filtrage $p_\theta(h_s|x_{0:s})$ pour tout $s \in [\![0, t]\!]$. Plus précisément, dans le modèle d'état linéaire et gaussien, la loi de filtrage $p_\theta(h_s|x_{0:s})$ et la loi $p_\theta(h_s|x_{0:s-1})$ sont des lois normales de paramètres respectifs $(\mu_{s|s}, \Sigma_{s|s})$ et $(\mu_{s|s-1}, \Sigma_{s|s-1})$:

$$p_\theta(h_s|x_{0:s}) = \mathcal{N}(h_s; \mu_{s|s}; \Sigma_{s|s}), \tag{2.29}$$

$$p_\theta(h_s|x_{0:s-1}) = \mathcal{N}(h_s; \mu_{s|s-1}; \Sigma_{s|s-1}). \tag{2.30}$$

Le filtre de Kalman propage alors récursivement ces paramètres :

$$\begin{aligned} \begin{bmatrix} \mu_{s|s} \\ \mu_{s+1|s} \end{bmatrix} &= \begin{bmatrix} \mu_{s|s-1} \\ F_s \mu_{s|s-1} \end{bmatrix} \\ &+ \begin{bmatrix} \Sigma_{s|s-1} G_s^T \\ F_s \Sigma_{s|s-1} G_s^T \end{bmatrix} \underbrace{(G_s \Sigma_{s|s-1} G_s^T + R_s)}_{\overline{L}_s}{}^{-1} \underbrace{(x_s - G_s \mu_{s|s-1})}_{\overline{x}_s}, \end{aligned} \tag{2.31}$$

$$\begin{bmatrix} \Sigma_{s|s} & \mathrm{Cov}(h_s, h_{s+1}|x_{0:s}) \\ (\ \mathrm{Cov}(h_s, h_{s+1}|x_{0:s}))^T & \Sigma_{s+1|s} \end{bmatrix} = \begin{bmatrix} \Sigma_{s|s-1} & \Sigma_{s|s-1}F_s^T \\ F_s\Sigma_{s|s-1} & F_s\Sigma_{s|s-1}F_s^T + Q_s \end{bmatrix} - \begin{bmatrix} \Sigma_{s|s-1}G_s^T \\ F_s\Sigma_{s|s-1}G_s^T \end{bmatrix} \overline{L}_s^{-1} \left[G_s\Sigma_{s|s-1}, G_s\Sigma_{s|s-1}F_s^T \right]. \tag{2.32}$$

Notamment, les quantités

$$\overline{x}_s = x_s - G_s\mu_{s|s-1}, \tag{2.33}$$
$$\overline{L}_s = G_s\Sigma_{s|s-1}G_s^T + R_s, \tag{2.34}$$

correspondent respectivement à la moyenne et la variance de la vraisemblance prédictive $p(x_s|x_{0:s-1})$.

Comme pour la loi de filtrage, on montre que la loi $p_\theta(h_{s-1}, h_s|x_{0:t})$ (voir équation (2.23)) ainsi que la loi de lissage $p_\theta(h_s|x_{0:t})$ sont des lois normales dont les paramètres (moyennes et covariances) sont calculés récursivement par propagation arrière [Shumway and Stoffer, 1982], ce qui permet une implémentation efficace de l'algorithme EM.

Cas 2 : états continus (cas général)

Dans le cas général (transitions non linéaires et/ou bruit non gaussien), le problème du filtrage ne peut plus être résolu de manière exacte. Il existe cependant plusieurs façons approchées de répondre à ce problème. Dans le cas où les transitions caractérisées par les équations (2.26) et (2.27) ne sont plus linéaires, il est possible de linéariser celles-ci grâce à un développement de Taylor au premier ordre. On peut alors utiliser un *filtre de Kalman étendu*, c'est-à-dire un filtre de Kalman dans ce modèle linéarisé [Chen, 1993, Anderson and Moore, 2012]. Par ailleurs, il est également possible de ne considérer que les moments d'ordre 1 et 2 des lois d'intérêt. Une approximation de ces moments peut alors être propagée par une transformation sans parfum. L'application récursive de cette transformation est appelée filtre de Kalman sans parfum [Julier and Uhlmann, 2004, Julier et al., 2000, Menegaz et al., 2015].

Une autre classe d'approximation est constituée des méthodes de filtrage particulaire ou méthodes de Monte-Carlo séquentielles [Doucet et al., 2001, Arulampalam et al., 2002, Hammersley, 2013]. Ces méthodes consistent à propager une approximation discrète de $p_\theta(h_{0:t}|x_{0:t})$ à l'aide d'un mécanisme d'échantillonnage d'importance avec ré-échantillonnage [Kahn and Marshall, 1953] :

$$\hat{p}_\theta(h_{0:t}|x_{0:t}) = \sum_{i=1}^{N} w_t^{(i)} \delta_{h_{0:t}^{(i)}}(h_{0:t}), \tag{2.35}$$

où δ représente la mesure de Dirac et les termes $w_t^{(i)}$ correspondent à un ensemble normalisé de poids. Ce mécanisme est basé sur une loi d'importance conditionnelle $q(h_t|h_{t-1})$. Plus précisément, la propagation des trajectoires pondérées $\left(h_{0:t}^{(i)}, w_t^{(i)}\right)_{i\in[\![1,N]\!]}$ se fait en trois étapes. Pour tout $s \in \mathbb{N}$, pour tout $i \in [\![1 : N]\!]$, une particule i est d'abord tirée selon q :

$$\tilde{h}_{s+1}^{(i)} \sim q(h_{s+1}|h_s^{(i)}). \tag{2.36}$$

On calcule ensuite son poids non normalisé

$$\tilde{w}_{s+1}^{u,(i)} = w_s^{(i)} \frac{p_\theta\left(\tilde{h}_{s+1}^{(i)}|h_s^{(i)}\right) p_\theta\left(x_{s+1}|\tilde{h}_{s+1}^{(i)}\right)}{q\left(\tilde{h}_{s+1}^{(i)}|h_s^{(i)}\right)}, \tag{2.37}$$

que l'on normalise ensuite en

$$\tilde{w}_{s+1}^{(i)} = \frac{\tilde{w}_{s+1}^{u,(i)}}{\sum_{j=1}^{N} \tilde{w}_{s+1}^{u,(j)}}. \tag{2.38}$$

Enfin, les trajectoires peuvent être ré-échantillonnées :

$$h_{0:s+1}^{(i)} \sim \sum_{j=1}^{N} \tilde{w}_{s+1}^{(j)} \delta_{(h_{0:s}^{(j)}, \tilde{h}_{s+1}^{(j)})}(h_{0:s+1}), \tag{2.39}$$

dont les nouveaux poids sont

$$w_{s+1}^{(i)} = \frac{1}{N}. \tag{2.40}$$

Cette étape de ré-échantillonnage, bien qu'optionnelle, permet de maintenir une plus grande

proportion de trajectoires de poids fort parmi toutes les trajectoires, au détriment de celles de poids faible. Si les trajectoires ne sont pas ré-échantillonnées, alors $\tilde{h}_{s+1}^{(i)}$ (respectivement $\tilde{w}_{s+1}^{(i)}$) devient simplement $h_{s+1}^{(i)}$ (respectivement $w_{s+1}^{(i)}$). Avec ou sans ré-échantillonnage, la procédure est répétée à partir de l'équation (2.36).

Cet algorithme fournit par ailleurs une approximation de la vraisemblance prédictive à partir des poids non normalisés (calculés dans l'équation (2.37)) :

$$\hat{p}_\theta(x_t|x_{0:t-1}) = \frac{1}{N}\sum_{i=1}^{N} \tilde{w}_t^{u,(i)}. \tag{2.41}$$

Une estimation de la vraisemblance est alors obtenue au moyen de l'équation (2.12). Enfin, $\hat{p}_\theta(h_{0:t}|x_{0:t})$ fournit également une approximation de (2.22) :

$$\hat{\mathbb{E}}_{\theta'}\left[\log p_\theta(h_{0:t}, x_{0:t})|x_{0:t}\right] = \sum_{i=1}^{N} w_{\theta',t}^{(i)} \log \hat{p}_\theta(h_{0:t}^{(i)}, x_{0:t}), \tag{2.42}$$

qu'il reste ensuite à maximiser. Ici, la notation $w_{\theta',t}^{(i)}$ rappelle simplement qu'il s'agit de poids construits à partir du paramètre θ' (voir (2.37)).

D'un point de vue pratique, l'approximation $\hat{p}$ peut être pauvre (en particulier, lorsque N est très petit devant t). Cette approximation peut néanmoins être raffinée par l'utilisation d'algorithmes de lissage particulaire [Kantas et al., 2015, Briers et al., 2010, Carvalho et al., 2010, Fearnhead et al., 2010] visant à améliorer l'approximation de $p(h_{s-1}, h_s|x_{0:t})$ dans (2.23).

Cas 3 : états latents discrets

De même que les systèmes d'état linéaires et gaussiens (cas 1) sont apparus dans les années 1960, les modèles de Markov cachés à états latents discrets sont apparus à la même époque [Baum and Petrie, 1966, Baum and Eagon, 1967, Forney, 1973] et ont été utilisés depuis dans des domaines tels que le traitement automatique du langage [Rabiner and Juang, 1986, Rabiner, 1989], la bio-informatique [Koski, 2001] ou les communications numériques [Viterbi, 1967, Forney, 1973].

Dans le cas discret, le problème est qu'il n'est pas possible de calculer directement la

vraisemblance par marginalisation de la loi jointe

$$p_\theta(x_{0:t}) = \sum_{h_{0:t}} p_\theta(h_{0:t}, x_{0:t}). \tag{2.43}$$

En effet, calculer (2.43) nécessite de sommer sur toutes les *suites* d'états latents possibles, d'où une explosion combinatoire. Par exemple, si l'espace des états latents contient K classes alors la somme porte sur K^{t+1} termes.

Le succès des HMM provient de ce que le calcul de la vraisemblance $p_\theta(x_{0:t})$ peut être réalisé en temps linéaire. En effet, la vraisemblance $p_\theta(x_{0:t})$ peut être vue comme la marginale de la loi $p_\theta(h_t, x_{0:t})$:

$$p_\theta(x_{0:t}) = \sum_{h_t} \alpha(h_t), \tag{2.44}$$

où

$$\alpha(h_s) = p_\theta(h_s, x_{0:s}) \tag{2.45}$$

correspond aux variables *forward*. Ces fonctions $\alpha(h_s)$ peuvent être calculées récursivement en temps linéaire dans le sens direct du temps (d'où le nom *forward*) :

$$\alpha(h_0) = p_\theta(h_0)p_\theta(x_0|h_0) \tag{2.46}$$

$$\alpha(h_{s+1}) = p_\theta(x_{s+1}|h_{s+1}) \sum_{h_s} p_\theta(h_{s+1}|h_s)\alpha(h_s). \tag{2.47}$$

Notons que $\alpha(h_s)$ est, à une constante près, la loi de filtrage

$$p_\theta(h_s|x_{0:s}) = \frac{\alpha(h_s)}{p_\theta(x_{0:s})}, \tag{2.48}$$

et l'équation (2.47) est l'analogue de l'équation (2.21). La vraisemblance prédictive est quant à elle

$$p(x_{s+1}|x_{0:s}) = \frac{\sum_{h_{s+1}} p_\theta(x_{s+1}|h_{s+1}) \sum_{h_s} p(h_{s+1}|h_s)\alpha(h_s)}{\sum_{h_s} \alpha(h_s)}. \tag{2.49}$$

D'après l'équation (2.23) (dans le cas discret, les intégrales deviennent des sommes), l'estimation des paramètres par l'algorithme EM requiert le calcul de la loi $p_\theta(h_{s-1}, h_s|x_{0:t})$

pour tout $s \in [\![0 : t]\!]$. À un terme de normalisation près, cette quantité est proportionnelle à

$$p_\theta(h_{s-1}, h_s, x_{0:t}) = \underbrace{p_\theta(x_{s+1:t}|h_s)}_{\beta(h_s)} \alpha(h_{s-1}) p_\theta(h_s|h_{s-1}) p_\theta(x_s|h_s). \tag{2.50}$$

En particulier, la loi $p_\theta(h_{s-1}, h_s, x_{0:t})$ dépend des variables *backward*

$$\beta(h_s) = p_\theta(x_{s+1:t}|h_s). \tag{2.51}$$

Tout comme les variables *forward*, ces variables peuvent être calculées récursivement en temps linéaire, dans le sens rétrograde du temps (d'où le nom *backward*) :

$$\beta(h_t) = 1 \tag{2.52}$$

$$\beta(h_s) = \sum_{h_{s+1}} \beta(h_{s+1}) p_{\theta_k}(x_{s+1}|h_{s+1}) p_\theta(h_{s+1}|h_s). \tag{2.53}$$

Le calcul récursif des fonctions $\alpha(h_s)$ et $\beta(h_s)$, pour tout s, est appelé *algorithme forward-backward*. Finalement, grâce à l'équation (2.50), nous obtenons

$$p_\theta(h_{s-1}, h_s|x_{0:t}) = \frac{\beta(h_s)\alpha(h_{s-1}) p_\theta(h_s|h_{s-1}) p_\theta(x_s|h_s)}{\sum_{h_{s-1}} \sum_{h_s} \beta(h_s)\alpha(h_{s-1}) p_\theta(h_s|h_{s-1}) p_\theta(x_s|h_s)}. \tag{2.54}$$

Il reste alors à maximiser par rapport aux paramètres d'intérêt que sont les probabilités de transitions $p_\theta(h_t|h_{t-1})$ et les probabilités $p_\theta(x_t|h_t)$ dont ce dernier calcul dépend de la forme de la loi (gaussienne, mélange de gaussiennes, etc.). La mise à jour de ces probabilités lors de l'étape M de l'algorithme EM est ainsi obtenue par le calcul de $p_\theta(h_{s-1}, h_s, x_{0:t})$. Cette version de l'algorithme EM appliquée aux HMM à états latents discrets est appelée algorithme de Baum-Welch [Baum et al., 1970, Rabiner, 1989].

Observons enfin que l'algorithme *forward-backward* permet, à θ fixé, de calculer la loi de lissage. En effet, l'expression de cette loi peut être vue comme une conséquence de l'équation (2.54) :

$$p_\theta(h_s|x_{0:t}) = \frac{\alpha(h_s)\beta(h_s)}{\sum_{h_s} \alpha(h_s)\beta(h_s)}. \tag{2.55}$$

2.3 Architectures neuronales

Les architectures neuronales sont des classes de fonctions très versatiles [Goodfellow et al., 2016] aux applications nombreuses et variées telles que le traitement du langage [Mikolov et al., 2011] ou l'analyse d'images [Krizhevsky et al., 2012].

2.3.1 Introduction aux réseaux de neurones

Un réseau de neurones est une combinaison successive de fonctions paramétrées appelées neurones. Un neurone commence par réaliser une transformation linéaire de l'entrée $\boldsymbol{x}$ par un vecteur de *poids* $\boldsymbol{w}$ et un *biais* scalaire b, puis applique une fonction non linéaire σ appelée *fonction d'activation* :

$$\boldsymbol{x} \mapsto \sigma(\boldsymbol{w}\boldsymbol{x} + b). \tag{2.56}$$

Parmi les fonctions d'activation les plus utilisées, nous pouvons par exemple citer la sigmoïde $x \mapsto \frac{1}{1+\exp -x}$, la tangente hyperbolique ou encore la fonction ReLu (pour *Rectified Linear Unit*) $x \mapsto \max(0, x)$. Plusieurs neurones peuvent être utilisés en parallèle et former une *couche* de neurones. La fonction obtenue est alors

$$\boldsymbol{x} \mapsto \sigma(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}), \tag{2.57}$$

où $\boldsymbol{W}$ est une matrice de poids, $\boldsymbol{b}$ un vecteur de biais et σ une fonction non linéaire appliquée terme à terme. Ces couches peuvent enfin être mises en série pour obtenir des classes de fonctions toujours plus complexes. Notamment, plusieurs théorèmes d'approximation universelle ont été formulés au sujet des réseaux de neurones [Cybenko, 1989, Hornik, 1991, Pinkus, 1999, Lu et al., 2017]. Ainsi, pour toute fonction réelle continue f (en dimension quelconque), il existe un réseau de neurones à une couche f_θ arbitrairement proche de cette dernière, pourvu que la fonction d'activation ne soit pas polynomiale [Pinkus, 1999]. Des résultats similaires ont été proposés pour les réseaux à plusieurs couches. Ainsi, toute fonction $f : \mathbb{R}^n \to \mathbb{R}$ Lebesgue-intégrable peut être approchée par un réseau dont les couches possèdent au moins $n+4$ neurones et dont les fonctions d'activations sont des ReLu pourvu que le réseau soit suffisamment profond [Lu et al., 2017]. Le nombre de couches, les nombres de neurones par couches et les fonctions d'activation sont des hyperparamètres qui caractérisent l'architecture du réseau de neurones et les poids et les biais sont quant à eux

des paramètres du modèle, appris *a posteriori* selon un jeu de données d'entraînement. Si l'on considère un modèle neuronal f_θ, un jeu de données d'apprentissage $\mathcal{E}$ (voir équation (2.4)) et une fonction de coût $L : \mathbb{R}^2 \to \mathbb{R}^+$, l'apprentissage d'un réseau de neurones est alors le problème d'optimisation visant à trouver le paramètre $\theta_n^\star$ qui minimise la somme $\sum_{i=1}^n L(f_\theta(x_i), y_i)$ (voir équation (2.7)). Ce problème d'optimisation est généralement traité par la méthode de descente de gradient. Cette dernière repose notamment sur le calcul de la dérivée

$$\frac{\partial L(f_\theta(x), y)}{\partial \theta}, \tag{2.58}$$

c'est-à-dire des dérivées partielles en tous les poids et les biais du réseau de neurones. Du fait de la structure en couches d'un réseau de neurones, le calcul de ces dérivées partielles peut être fait itérativement en commençant par les paramètres de la dernière couche puis en remontant couche par couche vers celle en entrée. Cette façon de calculer le gradient est appelée *rétropropagation du gradient* [Rumelhart et al., 1985, Hecht-Nielsen, 1992].

Toutefois, l'entrée d'un réseau de neurones comme nous l'avons décrit jusqu'à présent est de taille fixe (fonctionnement non récursif), ce qui semble peu compatible avec la modélisation de séries temporelles dont les observations s'accumulent. Une solution immédiate pourrait être de considérer une fenêtre glissante des observations, mais dans ce cas, la prédiction ne serait faite qu'à partir des éléments contenus dans la fenêtre sans porter sur l'intégralité des observations.

2.3.2 Réseaux de neurones récurrents

Prendre en compte l'historique des observations $x_{0:t}$ pour prédire x_{t+1} est une contrainte importante pour la prédiction de données temporelles. À cette fin, un réseau de neurones récurrent (*Recurrent Neural Network* ou RNN) introduit une variable latente h_t, fonction déterministe de toutes les observations $x_{0:t}$. Cette variable permet de résumer l'ensemble des observations $x_{0:t}$ dans un objet de taille fixe et être ainsi assimilé à une forme de mémoire. La mise à jour de cette mémoire est réalisée en ligne, observation après observation : pour tout t, à la réception d'une nouvelle observation x_{t+1}, le nouvel état latent h_{t+1} est calculé à l'aide d'une couche neuronale $h_{t+1} = f_{\theta_f}(h_t, x_{t+1})$. Finalement, à chaque instant t, une prédiction $\hat{x}_{t+1}$ de x_{t+1} peut être formulée à partir de la variable latente h_{t+1} selon un réseau de neurones $\hat{x}_{t+1} = g_{\theta_g}(h_{t+1})$. Le fonctionnement d'un réseau récurrent est illustré

par la figure 2.2. La boucle de rétroaction correspondant au maintien du vecteur latent (en bleu) a été dépliée dans le temps. Des prédictions (en rouge) peuvent être faites à chaque instant. Toutefois, selon les applications, la prédiction peut être réalisée après la lecture d'une suite d'observations (en vert). Dans notre cas, nous cherchons à prédire la prochaine observation x_{t+1} après avoir observé x_t pour tous t. Les prédictions d'un RNN peuvent ainsi être réinjectées en guise d'observations, ce qui permet la génération automatique de suites : il est alors envisageable de prédire des suites d'observations plutôt qu'une seule observation.

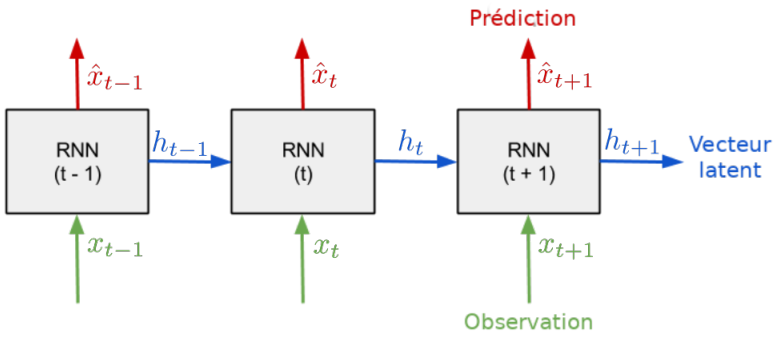


FIGURE 2.2 – Principe d'un réseau de neurones récurrent. À la réception d'une nouvelle observation (en vert) et étant donné un vecteur latent (en bleu), le réseau met à jour son vecteur latent et produit une prédiction (en rouge). Le vecteur latent mis à jour est utilisé au moment de la prochaine observation.

L'algorithme de rétropropagation du gradient peut encore être utilisé pour l'apprentissage des paramètres par descente de gradient. À la différence d'un réseau de neurones à propagation avant, il peut y avoir autant de gradients calculés que d'observations pour un même paramètre. En effet, la prédiction faite par le réseau récurrent de x_{t+1} étant donné $x_{0:t}$ et un vecteur h_0 peut être exprimée par

$$\Big(g_{\theta_g} \circ f_{\theta_f}(., x_t) \circ ... \circ f_{\theta_f}(., x_0)\Big)\,(h_0) = \hat{x}_{t+1}. \tag{2.59}$$

Cette écriture correspond à une vision « dépliée dans le temps » du réseau récurrent (voir figure 2.2). Dans ce cas, un paramètre est mis à jour selon la somme des dérivées partielles calculées à chaque instant. Cette adaptation de l'algorithme de rétropropagation est appelée *rétropropagation à travers le temps* [Robinson and Fallside, 1987, Werbos, 1990, Mozer, 1995].

En pratique, les gradients calculés pour un même paramètre tendent géométriquement vers 0 ou l'infini lorsque l'on remonte dans le passé. Ces phénomènes sont respectivement appelés *disparition du gradient* et *explosion du gradient*. L'explosion du gradient est généralement due à la multiplication répétée de poids de grande taille. L'apprentissage du réseau est alors particulièrement instable. Un moyen rapide de limiter ce comportement est de borner les valeurs prises par le gradient [Goodfellow et al., 2016, Goldberg, 2017]. Il est également possible d'intégrer un terme de régularisation à la fonction de coût en pénalisant les poids de trop grandes tailles [Pascanu et al., 2013]. Au contraire, la disparition du gradient est due à la multiplication répétée de poids de petite taille ainsi qu'à la combinaison répétée de fonctions d'activation dont les dérivées ont une valeur absolue inférieure à 1 (*e.g.* la sigmoïde). Dans ce cas, les observations les plus anciennes (relativement à l'indice courant) sont ignorées lors de l'apprentissage. De fait, il est difficile d'apprendre des dépendances à long terme. Pour y pallier, des architectures plus sophistiquées ont été proposées telles que les *Long Short Term Memories* (LSTM) et les *Gated Recurrent Units* (GRU) [Chung et al., 2014]. En particulier, LSTM et GRU complexifient la boucle de retour du RNN classique pour atténuer le phénomène de disparition du gradient. Ces modèles sont généralement privilégiés en pratique bien que leur étude théorique soit sensiblement plus difficile.

RNN génératifs

Dans le cas de la prédiction des observations à venir, une approche probabiliste paraît plus pertinente lorsque l'on souhaite quantifier l'incertitude liée à notre prédiction. En effet, considérer une loi de probabilité sur les observations fournit une représentation plus riche du futur qu'un simple mode donné par une approche déterministe. Par exemple, chercher à exprimer x_{t+1} comme la réalisation d'une loi normale de moyenne et variance $(\mu_{t+1}, \sigma_{t+1})$ permet non seulement de prédire $x_{t+1} = \mu_{t+1}$ mais aussi de donner un intervalle de confiance grâce à σ_{t+1}. Il est ainsi possible de construire une loi sur les observations à partir d'un RNN et en faire ainsi un modèle génératif. Pour cela, il suffit de remplacer g_{θ_g} par une loi paramétrée $p_{\theta_g}(x_t|h_t)$. On obtient ainsi une famille de modèles pour p :

$$\begin{aligned} p_\theta(x_{0:t}) &= p_\theta(x_0) \prod_{s=0}^{t-1} p_\theta(x_{s+1}|x_{0:s}) \,; \text{ (voir équation (2.9))} \\ &\overset{\text{RNN}}{=} p_{\theta_g}(x_0) \prod_{s=0}^{t-1} p_{\theta_g}(x_{s+1}|h_s), \end{aligned} \tag{2.60}$$

où $\theta = (\theta_f, \theta_g)$ et h_s est fonction de $x_{0:s}$:

$$h_s = f_{\theta_f}(h_{s-1}, x_s). \tag{2.61}$$

Par construction, du fait de la mise à jour déterministe des états latents décrite dans (2.61), évaluer la suite des états latents peut être fait de manière immédiate et exacte, ce qui n'est pas le cas dans une chaîne de Markov cachée. Ainsi, l'accès à la loi $p_\theta(x_{t+1}|x_{0:t})$ pour tout x_{t+1} est aisé. L'estimation des paramètres θ se fait naturellement en maximisant la log-vraisemblance $\log(p_\theta(x_{0:t})) = \sum_t \log(p_\theta(x_t|h_t = h(x_{0:t}))$ par la rétropropagation classique.

2.4 Conclusion

À travers ce chapitre, nous avons pu distinguer deux grandes classes de modèles pour la prédiction de données temporelles : les modèles de Markov cachés d'un côté et les réseaux de neurones récurrents de l'autre. À première vue, ces modèles sont différents. Par exemple, un RNN présente des transitions déterministes que ne possède pas un HMM. Toutefois, malgré leurs différences, ils offrent tous deux un point de vue génératif sur les observations. De plus, pour les deux modèles, ces dernières sont modélisées par l'intermédiaire de variables latentes. Si nous avons pu dans ce chapitre décrire comment l'inférence et l'apprentissage de ces modèles pouvaient être réalisés, une question naturelle est de comparer les lois sur les observations $p_{\theta_{\text{HMM}}}(x_{0:t})$ et $p_{\theta_{\text{RNN}}}(x_{0:t})$ offertes par chacun des deux modèles. Une telle comparaison peut s'avérer difficile dans les cas les plus généraux. En particulier, nous ne traiterons pas le cas des LSTM et des GRU. Nous montrons dans le chapitre 6 que sous des hypothèses raisonnables, une analyse comparative théorique est possible. Pour cela, nous introduirons un modèle appelé *Modèle Génératif Unifié* (GUM) englobant les deux architectures.

Chapitre 3

Stockage des motifs de causalité observés dans un log

Dans ce chapitre, nous proposons une représentation causale d'un log d'alarmes. En particulier, nous souhaitons stocker grâce à cette représentation les motifs de causalités basées sur les caractéristiques spatiales (machine ou fonctionnalités impactées) et temporelles (date de début et de fin) des alarmes, et, éventuellement, les connaissances *a priori* de l'expert. Du fait de ces caractéristiques, la représentation causale d'un log requiert une structure plus complexe que celle d'une suite temporelle. Tout d'abord, nous lions les événements d'un log par une relation de causalités potentielles. Cela nous permet de modéliser le log en une structure de graphe causale. L'ensemble des chaînes de causalité potentielle observées sont alors stockées dans une structure appelée DIG-DAG dont la construction est inspirée de l'algorithme d'Ukkonen [Ukkonen, 1995]. Ces deux travaux construisent une structure de données en traitant en ligne une chaîne de caractères : alors que l'algorithme d'Ukkonen construit un arbre suffixe, notre proposition construit un DAG proche d'un automate fini déterministe. La structure obtenue se veut interprétable afin d'être utilisée par un opérateur humain pour la gestion de réseaux. Le fait que la construction soit en ligne permet notamment l'utilisation de cette structure pour la prédiction d'alarmes. L'ensemble des travaux présentés dans ce chapitre ont été présentés à la conférence CNSM [Bouillard et al., 2018].

Dans la section 3.1, nous introduisons une relation de causalité potentielle entre les événements d'un fichier de trace. Puis, dans la section 3.2, nous stockons à la volée l'ensemble des motifs de causalité observés dans une structure de graphe appelée DIG-DAG. Enfin, la

section 3.3 propose diverses variantes et extensions.

3.1 Représentation d'un log d'alarmes

Afin d'administrer un réseau, une pratique essentielle est de consigner tout événement dans un fichier de trace, ou *log d'alarmes*. Un tel fichier peut alors être utilisé pour suivre le bon fonctionnement du réseau, pour identifier efficacement la cause racine d'une panne, ou encore prédire des pannes futures.

Un log d'alarmes est une collection d'événements classés dans l'ordre chronologique (voir exemple 3). Notre but est de trouver toutes les chaînes de causalité potentielle entre les événements à partir de plusieurs logs d'alarmes, traités en ligne. Cette section est dédiée à la représentation d'un log à travers la notion d'événements et la définition d'une relation de causalité entre ces derniers.

Exemple 3 *Ci-dessous un extrait de log réel. Cet extrait a été anonymisé et chaque ligne mise sous le format* `((machine, alarme, severité), activation | date)` *:*

```
((m_143, a_25, 2), fin, 2019-02-11 01:36:22)
((m_271, a_18, 3), fin, 2019-02-11 01:36:36)
((m_271, a_17, 3), fin, 2019-02-11 01:36:36)
((m_271, a_18, 3), début, 2019-02-11 01:36:37)
((m_271, a_17, 3), début, 2019-02-11 01:36:38)
((m_1115, a_24, 4), fin, 2019-02-11 01:36:54)
((m_268, a_26, 1), fin, 2019-02-11 01:37:03)
((m_268, a_26, 1), début, 2019-02-11 01:37:38)
((m_720, a_25, 2), fin, 2019-02-11 01:38:04)
((m_16, a_22, 2), fin, 2019-02-11 01:39:46)
((m_308, a_24, 4), début, 2019-02-11 01:39:54)
((m_514, a_7, 3), début, 2019-02-11 01:39:59)
((m_271, a_13, 3), début, 2019-02-11 01:40:27)
((m_571, a_23, 2), fin, 2019-02-11 01:40:43)
((m_571, a_24, 3), fin, 2019-02-11 01:40:46)
```

Même après en avoir simplifié la forme (certaines entrées du log sont écartées), la lecture et l'exploitation d'un log restent difficiles, surtout si le log en question est de grande taille.

Nous présentons la notion d'événement dans le paragraphe 3.1.1. Puis, dans le paragraphe 3.1.2, nous définissons une relation de *causalité potentielle* entre deux événements. Dans le paragraphe 3.1.3, nous donnons deux représentations d'un log d'alarmes : une première représentation à la fois graphique et intuitive, appelée *graphe orienté d'intervalle* (DIG), et une seconde sous forme de mot, plus pratique pour une utilisation à la volée. Enfin, dans le paragraphe 3.1.4, nous définissons formellement l'ensemble des chaînes de causalité à stocker.

Remarque générale sur les notations employées

Dans les chapitres 3 à 5, nous utiliserons les notations de la théorie des langages. Étant donné un ensemble fini Σ, nous notons Σ^* l'ensemble des suites finies sur Σ appelées *mots* ; en particulier, ε définit le mot vide. Si w est un mot et $\sigma \in \Sigma$ une lettre, $|w|_\sigma$ est le nombre d'occurrences de σ dans w. Le préfixe de w de longueur $k \in \mathbb{N}$ est noté $w_{\leq k}$.

3.1.1 Collection d'événements

Un événement $e = (\sigma, [s, t])$ est défini par trois paramètres :

- l'intervalle $[s, t]$ (avec $s, t \in \mathbb{N}$ et $s < t$) décrit la temporalité de l'événement. Plus précisément, il est *actif* de la date s à la date t.
- le symbole σ fournit quant à lui une description spatiale de l'événement. Mathématiquement, un symbole ne porte pas de sémantique particulière ; en pratique, l'expert choisit les informations non temporelles qui caractérisent le mieux un événement. Un événement peut par exemple indiquer quelle est la machine ou encore quelle est l'alarme correspondante (une utilisation CPU trop forte, un espace de stockage insuffisant, etc.). La notion d'« espace » fait référence, par un léger abus de langage, à tout ce qui n'est pas temporel. Il faut donc y voir un espace logique plus que physique. Enfin, l'ensemble de tous les symboles présents dans un log est noté Σ, notre *alphabet*.

Soit $\mathcal{E} = (e_i)_{i \in [\![1;n]\!]}$ un log d'alarmes. Pour tout $i \in [\![1;n]\!]$, nous écrivons $e_i = (\sigma_i, [s_i, t_i])$. Afin de garantir l'unicité et la cohérence d'un log étant donné une suite d'événements, nous faisons les hypothèses suivantes :

(H1) L'ensemble des dates de début et de fin du log sont toutes distinctes.

(H2) Deux événements portant le même symbole ne peuvent être co-occurrents : $\forall i, j \in [\![1;n]\!], [s_i, t_i] \cap [s_j, t_j] \neq \emptyset \Rightarrow \sigma_i \neq \sigma_j \vee i = j$.

Ces hypothèses ne sont pas restrictives : il est possible de déterminer une relation d'ordre total sur les dates d'émission à partir de l'ordre dans lesquels les événements sont renseignés dans le log. Si deux événements portant le même symbole sont co-occurrents, il est alors possible pour respecter (H2) de les agréger en un seul événement dont l'intervalle d'activité est l'union des intervalles d'activité. Autrement dit, si $[s_i, t_i] \cap [s_j, t_j] \neq \emptyset$, $e_i = (\sigma, [s_i, t_i])$ et $e_j = (\sigma, [s_j, t_j])$ peuvent être remplacés par $(\sigma, [\min(s_i, s_j), \max(t_i, t_j)])$. L'hypothèse (H2) évite notamment la présence d'informations ambiguës voire contradictoires : un événement ne peut pas recommencer s'il n'est pas encore terminé.

Exemple 4 *Le réseau jouet de la figure 3.1 implique trois machines. Un exemple d'événement est* $e = (c, [1, 3])$ *: le symbole* c *a été émis pendant l'intervalle* $[1, 3]$*. Notons que le symbole* c *encode le fait que l'événement* e *a été émis depuis la machine* m_1*.*

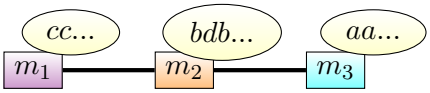


FIGURE 3.1 – Exemple jouet de réseau, qui peut être utilisé pour définir une relation spatiale de causalité.

3.1.2 Causalités spatio-temporelles

À partir des informations spatio-temporelles, il est possible de définir une relation de causalité potentielle entre deux événements. Cette relation de causalité est définie à partir de deux relations distinctes : l'une basée uniquement sur les informations temporelles des événements, l'autre reposant uniquement sur les informations spatiales contenues dans le symbole.

La relation temporelle $\mathcal{R}_t$ définit la notion de causalité potentielle entre deux événements uniquement à partir des attributs temporels. Elle permet de définir une première notion de causalité potentielle sans connaissance *a priori* de la spatialité des événements considérés. Dans le cas où les événements possèdent une date de début et de fin, il y a relation temporelle de $e = (\sigma, [s, t])$ vers $e' = (\sigma', [s', t'])$ si l'événement e' apparaît pendant que e est actif. Plus formellement, $e\mathcal{R}_t e' \Leftrightarrow s' \in [s, t]$. Il est tout à fait possible de définir la relation temporelle différemment selon le format des données utilisées (des événements sans date de fin par exemple) ou si une autre définition paraît plus pertinente (dans un cas où la causalité pourrait avoir lieu même après l'interruption du premier événement par exemple).

La relation spatiale $\mathcal{R}_s$ quant à elle vient ajouter les considérations spatiales manquantes à la relation temporelle seule. Cette relation repose sur la définition *a priori* d'un ensemble $\mathcal{C} \subseteq \Sigma^2$ de relations spatiales valides de causalité potentielle :

$$e\mathcal{R}_s e' \Leftrightarrow \sigma\sigma' \in \mathcal{C}.$$

Par exemple, si deux événements sont émis par des machines voisines, il est raisonnable d'autoriser la relation spatiale correspondante dans $\mathcal{C}$. Ainsi, la définition de $\mathcal{C}$ permet l'incorporation de connaissances *a priori* par un expert. Si les alarmes renseignées dans σ et σ' sont notoirement peu corrélées, alors il est judicieux de retirer $\sigma\sigma'$ et $\sigma'\sigma$ de $\mathcal{C}$. La relation $\mathcal{C}$ n'est pas nécessairement symétrique. Enfin, l'hypothèse (H2) nous permet de rejeter tous les couples de la forme $\sigma\sigma$ de $\mathcal{C}$ sans perte de généralité.

Définition 1 (Cause potentielle) *Un événement e est une cause potentielle d'un événement e' si et seulement si $e\mathcal{R}_t e' \wedge e\mathcal{R}_s e'$. Dans ce cas, nous écrivons $e \rightarrow e'$.*

Exemple 5 *Supposons que la relation spatiale de causalité repose sur la topologie représentée dans la figure 3.1 : $\mathcal{C} = \Sigma^2 \setminus \{ac, ca\}$. Considérons deux événements $e = (a, [4, 7])$ et $e' = (c, [6, 10])$. Nous avons alors $e\mathcal{R}_t e'$ mais $\neg e\mathcal{R}_s e'$, ainsi $e \not\rightarrow e'$. L'ensemble $\mathcal{C}$ est illustré en figure 3.2.*

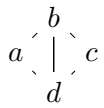


FIGURE 3.2 – Relations de causalités spatiales valides $\mathcal{C}$ du réseau jouet illustré en figure 3.1 : ac et ca sont interdites puisqu'il n'y a pas de lien direct entre les machines m_1 et m_3.

3.1.3 Deux façons de représenter un log

Graphe orienté d'intervalles (DIG)

Précédemment, nous avons défini la notion d'événement et une relation de causalité potentielle entre deux événements. Cela nous mène ainsi à une représentation graphique naturelle d'un log d'alarmes : soit $\mathcal{E}$ un log d'alarmes, et π_s une fonction d'étiquetage définie par $\pi_s(e) = \sigma$ où $e = (\sigma, [s, t]) \in \mathcal{E}$ et $\sigma \in \Sigma$. Le triplet $(\mathcal{E}, \rightarrow, \pi_s)$ définit un graphe orienté étiqueté dont les sommets sont les événements du log, étiquetés par leurs symboles et dont les arcs sont exactement les causalités potentielles. Nous appelons cette structure *graphe orienté d'intervalles* (DIG) d'après sa construction basée sur des intervalles de temps, et ce, bien que cette définition diffère légèrement des graphes d'intervalles classiques.

On remarquera que la relation temporelle rend le graphe acyclique. Par ailleurs, grâce aux hypothèses (H1) et (H2), le DIG d'un log d'alarmes est uniquement défini.

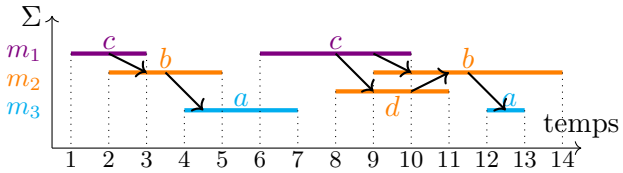


FIGURE 3.3 – Un exemple de DIG, les relations spatiales valides sont $\mathcal{C} = \Sigma^2 \backslash \{ac, ca\}$.

Exemple 6 *La figure 3.3 décrit le DIG correspondant à une collection de sept événements issus de la topologie représentée en figure 3.1. Le temps est représenté le long de l'axe des abscisses. Les sommets du DIG sont les segments correspondant chacun à un événement. Chaque événement correspond dans le DIG à un sommet qui est représenté en figure 3.3 par un segment horizontal conformément aux dates de début et de fin de l'événement correspondant. On peut vérifier que les arcs ont bien été définis à partir de $\mathcal{R}_t$ et $\mathcal{R}_s$ et qu'ils*

représentent une causalité potentielle entre deux événements. En particulier, l'arc (a, c) *a été écarté selon* $\mathcal{C} = \Sigma^2 \backslash \{ac, ca\}$*, illustrée dans la figure 3.2.*

Représentation en mot

Rappelons que nous visons à définir une structure de données qui stocke toutes les chaînes de causalité potentielle observées dans un log d'alarmes, et que nous souhaitons réaliser sa construction en temps réel. Il est donc naturel de représenter séparément le début et la fin de chaque événement. Dans ce but, nous introduisons l'alphabet $\overline{\Sigma}$, une copie de Σ disjointe. Chaque événement $e = (\sigma, [s, t])$ émet un symbole σ à la date s et un symbole $\overline{\sigma}$ à la date t. Le mot correspondant à un log $\mathcal{E}$ est le mot composé de tous les symboles émis dans leur ordre d'apparition.

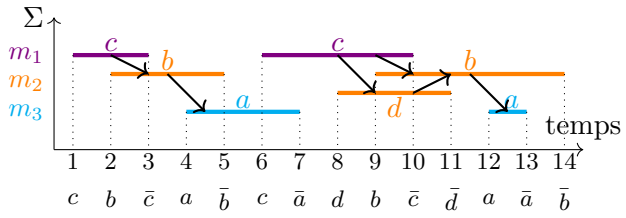


FIGURE 3.4 – Passage du DIG au mot.

Exemple 7 *Comme l'illustre la figure 3.4, le mot correspondant au DIG de la figure 3.3 est* $w = cb\bar{c}a\bar{b}c\bar{a}db\bar{c}\bar{d}a\bar{a}\bar{b}$*.*

Notons que ce mot est indépendant de la relation $\mathcal{C}$ et encode seulement la relation temporelle $\mathcal{R}_t$. D'après l'hypothèse (H1), ce mot est uniquement défini et possède les propriétés suivantes :

(P1) $\forall \sigma \in \Sigma, |w|_\sigma = |w|_{\overline{\sigma}}$;

(P2) $\forall j \in [\![1; |w|]\!], |w_{\leq j}|_{\overline{\sigma}} \leq |w_{\leq j}|_\sigma \leq |w_{\leq j}|_{\overline{\sigma}} + 1$.

Ces propriétés peuvent être vues comme garantissant un parenthésage correct du log : étant donné un certain symbole, chaque événement portant ce symbole commence puis finit (et non l'inverse). De plus, des alarmes portant le même symbole ne peuvent pas

se superposer, ce qui est décrit par l'inégalité de droite de la seconde propriété et une conséquence directe de l'hypothèse (H2).

Définition 2 (Mot correct) *Un mot est correct si et seulement s'il satisfait les propriétés (P1) et (P2). C'est un préfixe correct si et seulement s'il satisfait (P2). On note $\mathcal{L}$ le langage des préfixes corrects.*

Représenter un log par un mot correct plutôt que par sa forme brute ou son DIG facilite sa manipulation. Dans le cas où le log est découvert en temps réel, considérer des préfixes corrects offre un formalisme plus flexible puisqu'il autorise des événements qui seraient encore en cours. En effet, à tout instant, un événement peut être encore actif. Dans ce cas, la représentation sous forme de mot est plus pratique puisqu'elle décrit parfaitement la suite de début et de fin d'événements.

Remarquons que si w est un préfixe correct, alors tous ses préfixes le sont aussi. En réalité, en ordonnant les événements d'un log selon leurs dates de début, le mot correspondant est correct et caractérise complètement le log. Soit $\mathcal{E}$ une collection de n événements $e_i = (\sigma_i, [s_i, t_i])$ ordonnés par date de début ($i < j$ si et seulement si $s_i < s_j$) et tel que $\cup_{i=1}^{n}\{s_i, t_i\} = [\![1; 2n]\!]$. Le mot correspondant est noté $w(\mathcal{E}) = w_1...w_{2n}$ avec $w_{s_i} = \sigma_i$ et $w_{t_i} = \overline{\sigma}_i$. Réciproquement, si $w = w_1...w_{2n}$ est un mot correct, soit s_i l'indice correspondant à la i-ème occurrence d'un symbole de Σ, et $t_i = \inf\{t \geq s_i | w_t = \overline{w}_{s_i}\}$. Le mot w induit alors la collection d'événements $\mathcal{E}(w) = ((w_{s_i}, [s_i, t_i]))_{i \in [\![1;n]\!]}$. Il est possible de vérifier que $w(\mathcal{E}(w)) = w$ et $\mathcal{E}(w(\mathcal{E})) = \mathcal{E}$.

Le mot $w(\mathcal{E})$ correct associé à un log $\mathcal{E}$ ne permet pas à lui seul de caractériser la causalité potentielle d'un événement sur un autre. En effet, $w(\mathcal{E})$ n'encode pas les connaissances *a priori* contenues dans $\mathcal{C}$. Le lemme 1 montre que $w(\mathcal{E})$ et $\mathcal{C}$ suffisent à caractériser le DIG $(\mathcal{E}, \rightarrow, \pi_s)$:

Lemme 1 *En utilisant les notations introduites ci-dessus, soit $\mathcal{E}$ un log de n événements, w le mot correct associé et $(\mathcal{E}, \rightarrow, \pi_s)$ le DIG associé. Nous avons l'équivalence*

$$e_i \rightarrow e_j \Leftrightarrow s_j \in [s_i, t_i] \wedge w_{s_i} w_{s_j} \in \mathcal{C}.$$

Preuve. w étant le mot correct associé à $\mathcal{E}$, tout événement e_i s'écrit $e_i = (w_{s_i}, [s_i, t_i])$.

Il suffit alors d'appliquer les définitions des relations de causalité potentielle, spatiale et temporelle. □

3.1.4 Langage des mots à stocker

Nous cherchons à présent à définir tous les mots correspondants aux chemins du DIG. Le mot d'un chemin est la suite d'étiquettes portées par chaque sommet impliqué dans ce chemin. Dans le DIG-DAG de la figure 3.3, l'ensemble des mots à stocker est $\{\varepsilon, c, b, a, d, cb, ba, cd, db, cba, cdb, dba, cdba\}$.

La notion de *sous-mot admissible* caractérise le mot d'un chemin du DIG à partir de la représentation du log en mot :

Définition 3 (Sous-mots admissibles) *Soit $w = w_1...w_n \in \mathcal{L}$ un préfixe correct. La fonction $\phi : [\![1; k]\!] \to [\![1; n]\!]$ est une sous-suite admissible de w si et seulement si :*

- *ϕ est strictement croissante ;*
- *$w_{\phi(1)}w_{\phi(2)}...w_{\phi(k)} \in \Sigma^k$;*
- *$\forall j \in [\![1; k-1]\!], w_{\phi(j)}w_{\phi(j+1)} \in \mathcal{C}$;*
- *$\forall j \in [\![1; k-1]\!], |w_{\phi(j)}...w_{\phi(j+1)}|_{\overline{w}_{\phi(j)}} = 0$.*

Le mot $w_{\phi(1)}...w_{\phi(k)}$ est appelé sous-mot admissible de w, *et l'ensemble des sous-mots admissibles de w est noté $\mathcal{L}(w)$.*

La correspondance entre sous-mots admissibles et chemins étiquetés du DIG peut alors être formellement énoncée.

Théorème 1 *Soit $\mathcal{E}$ un log d'alarmes, et w un mot correct construit à partir de $\mathcal{E}$. Alors l'ensemble des chemins étiquetés de $(\mathcal{E}, \to, \pi_s)$ est exactement $\mathcal{L}(w)$.*

Preuve. Considérons un DIG $(\mathcal{E}, \to, \pi_s)$ et w son mot correct associé.

Soit $e_1...e_k$ un chemin étiqueté du DIG, montrons tout d'abord que $\pi_s(e_1)...\pi_s(e_k)$ correspond à un mot de $\mathcal{L}(w)$. Pour tout $i \in [\![1; k]\!], e_i = (w_{s_i}, [s_i, t_i])$. Nous notons ϕ la fonction qui à $i \in [\![1; k]\!]$ associe s_i. Alors :

- ϕ est une fonction strictement croissante car $\forall i \in [\![1; k-1]\!]$, $e_i \to e_{i+1} \Rightarrow e_i \mathcal{R}_t e_{i+1} \Rightarrow s_i < s_{i+1} \Rightarrow \phi(i) < \phi(i+1)$;
- $\pi_s(e_1)...\pi_s(e_k) = w_{\phi(1)}...w_{\phi(k)}$ ne correspond qu'à des débuts d'intervalles et appartient donc à Σ^k ;
- d'après le lemme 1, pour tout $i \in [\![1; k]\!]$, $e_i \to e_{i+1}$ donc la paire $w_{\phi(i)}w_{\phi(i+1)} \in \mathcal{C}$ est valide *a priori* ;
- mais aussi $\phi(i+1) < t_i$, c'est-à-dire $|w_{\phi(j)}...w_{\phi(j+1)}|_{\overline{w}_{\phi(j)}} = 0$.

Ainsi, ϕ est une sous-suite admissible et $\pi_s(e_1)...\pi_s(e_k) \in \mathcal{L}(w)$.

Réciproquement, notons ϕ la sous-suite admissible associée à un sous-mot admissible $w_{\phi(1)}...w_{\phi(k)} \in \mathcal{L}(w)$. Comme w est le mot correct associé à $\mathcal{E}$, chaque $w_{\phi(i)} \in \Sigma$ est associé à $e_i = (w_{\phi(i)}, [\phi_i, t_i])$. D'ailleurs, les k événements sont tous distincts puisque ϕ est strictement croissante. Or, pour tout $i \in [\![1; k-1]\!]$, $w_{\phi(i)}w_{\phi(i+1)} \in \mathcal{C}$ d'une part et $\phi(i) < \phi(i+1) < t_i$ d'autre part. Donc, d'après le lemme 1, $e_i \to e_{i+1}$. Nous considérons donc bien un chemin étiqueté du DIG. □

Construction récursive de $\mathcal{L}(w)$. Dans la définition 3, il est possible de distinguer deux types de suites admissibles : les sous-suites admissibles *prolongeables* d'une part et les sous-suites admissibles *non prolongeables* d'autre part. Intuitivement, les sous-suites prolongeables sont celles qui peuvent éventuellement être étendues à la lecture des prochains symboles de w. De même, on peut distinguer le sous-ensemble $\mathcal{L}^c(w) \subseteq \mathcal{L}(w)$ des mots prolongeables, c'est-à-dire des mots correspondant à au moins une suite prolongeable. Cette notion de mot prolongeable permet une construction récursive de $\mathcal{L}(w)$ qui sera utilisée dans la section 3.2.

Définition 4 (Sous-mot admissible prolongeable) *Soit $w \in \mathcal{L}$ un préfixe correct de taille n. Soit ϕ une sous-suite admissible telle que $\{w_{\phi(i)}\}_{i \in [\![1;k]\!]} \in \mathcal{L}(w)$ soit un sous-mot admissible de w de taille $k \leq n$.*

Ce sous-mot est prolongeable *si et seulement si $|w_{\phi(k)}...w_n|_{\overline{w}_{\phi(k)}} = 0$. L'ensemble des sous-mots prolongeables de w est noté $\mathcal{L}^c(w)$.*

Théorème 2 *Pour tout $w \in \mathcal{L}$ et $\sigma \in \Sigma$:*

- $\mathcal{L}(\varepsilon) = \mathcal{L}^c(\varepsilon) = \{\varepsilon\}$;

- *si* $w\sigma \in \mathcal{L}$, *alors* $\mathcal{L}^c(w\sigma) = \mathcal{L}^c(w) \cup (\mathcal{L}^c(w)\sigma \cap (\Sigma^*\mathcal{C} \cup \Sigma))$ *et* $\mathcal{L}(w\sigma) = \mathcal{L}(w) \cup (\mathcal{L}^c(w)\sigma \cap (\Sigma^*\mathcal{C} \cup \Sigma))$;
- *si* $w\overline{\sigma} \in \mathcal{L}$, $\mathcal{L}^c(w\overline{\sigma}) = \mathcal{L}^c(w) \backslash \Sigma^*\sigma$ *et* $\mathcal{L}(w\overline{\sigma}) = \mathcal{L}(w)$.

Preuve. Tout d'abord, $\{\varepsilon\} \subseteq \mathcal{L}^c(\varepsilon) \subseteq \mathcal{L}(\varepsilon) = \{\varepsilon\}$. Ce qui prouve la première assertion.

Puis, soient $w \in \mathcal{L}$ et $\sigma \in \Sigma$. Supposons que $w\sigma \in \mathcal{L}$. Le mot w est toujours prolongeable s'il est vide (σ est le premier symbole observé). Si w n'est pas vide, l'ensemble des sous-mots admissibles de $w\sigma$ peut être partitionné en deux ; ceux qui ont été construits seulement à partir de w d'une part et ceux qui terminent par σ d'autre part. Pour tout sous-mot appartenant à ce deuxième ensemble et de longueur k, il existe donc une sous-suite admissible ϕ telle que $(w\sigma)_{\phi(k)} = \sigma$. D'après la définition 3, $\{(w\sigma)_{\phi(i)}\}_{i\in[\![1;k-1]\!]} = \{w_{\phi(i)}\}_{i\in[\![1;k-1]\!]}$ est un sous-mot admissible de w. De plus, puisque $|(w\sigma)_{\phi(k-1)}...(w\sigma)_{\phi(k)}|_{\overline{w\sigma}_{\phi(k-1)}} = |w_{\phi(k-1)}...w_{|w|}\sigma|_{\overline{w}_{\phi(k-1)}} = 0$, alors $\{(w\sigma)_{\phi(i)}\}_{i\in[\![1;k-1]\!]}$ est un sous-mot admissible prolongeable. La définition 3 impose également que $w_{\phi(k-1)}\sigma \in \mathcal{C}$. Autrement dit, $\{(w\sigma)_{\phi(i)}\}_{i\in[\![1;k]\!]}$ doit également appartenir au langage $\Sigma^*\mathcal{C}$.

Réciproquement, on considère un mot $w' \in (\mathcal{L}^c(w)\sigma \cap (\Sigma^*\mathcal{C} \cup \Sigma))$. On note k sa longueur. Il existe une sous-suite admissible ϕ' telle que $w'_{\leq k-1} = \{w_{\phi'(i)}\}_{i\in[\![1;k-1]\!]}$. Il suffit alors de définir la sous-suite ϕ telle que $\phi(i) = \phi'(i)$ pour tout $i \in [\![1; k-1]\!]$ et $\phi(k) = \sigma$. Cette sous-suite vérifie les critères de la définition 3, ce qui permet de conclure : $\mathcal{L}(w\sigma) = \mathcal{L}(w) \cup (\mathcal{L}^c(w)\sigma \cap \Sigma^*\mathcal{C})$.

Comme σ est le dernier symbole reçu et qu'il n'appartient pas à $\overline{\Sigma}$, tout sous-mot admissible de $w\sigma$ finissant par σ est aussi prolongeable. Ainsi, $\mathcal{L}^c(w\sigma) = \mathcal{L}^c(w) \cup (\mathcal{L}^c(w)\sigma \cap \Sigma^*\mathcal{C})$.

Enfin, si maintenant $w\overline{\sigma} \in \mathcal{L}$. L'ensemble des sous-mots admissibles de w étant un sous-ensemble de Σ^*, l'ajout d'un symbole barré à w n'introduit pas de nouveau sous-mot admissible à ceux déjà présents dans $\mathcal{L}(w)$. Ainsi $\mathcal{L}(w\overline{\sigma}) = \mathcal{L}(w)$.

De la même manière, $\mathcal{L}^c(w\overline{\sigma}) \subseteq \mathcal{L}^c(w)$. Toutefois, pour tout sous-mot admissible prolongeable de w, d'après la définition 4, ce n'est plus un sous-mot prolongeable de $w\sigma$ s'il finit par le symbole σ. □

Exemple 8 *Reprenons le mot correct w défini dans l'exemple 7. En particulier, nous considérons $w_{\leq 8} = cb\bar{c}a\bar{b}c\bar{a}d$ les huit premiers symboles de ce mot. Nous avons* $\mathcal{L}(w_{\leq 8}) =$

$\{\varepsilon, a, b, c, d, ba, cb, cd, cba\}$ *et* $\mathcal{L}^c(w_{\leq 8}) = \{\varepsilon, c, d, cd\}$. *Le théorème 2 donne* $\mathcal{L}(w_{\leq 9}) = \mathcal{L}(w_{\leq 8}b) = \{\varepsilon, a, b, c, d, ba, cb, cd, db, cdb, cba\}$ *et* $\mathcal{L}^c(w_{\leq 9}) = \mathcal{L}^c(w_{\leq 8}b) = \{\varepsilon, b, c, d, cb, db, cd, cdb\}$.

3.2 Construction d'une structure de stockage

Notre but dans cette section est de définir une structure de données capable de stocker efficacement tous les sous-mots admissibles $\mathcal{L}(w)$ d'un préfixe correct w (voir définition 3). La construction doit se faire en ligne, en découvrant les symboles de w de gauche à droite, c'est-à-dire au fur et à mesure des activations et désactivations des événements du log.

Le paragraphe 3.2.1 présente une nouvelle structure de données appelée DIG-DAG. Nous donnons alors deux constructions possibles d'une telle structure : la plus simple d'une part, le DIG-trie, présentée dans le paragraphe 3.2.2 où la structure de graphe est un arbre et une version plus compacte d'autre part dans le paragraphe 3.2.3. Enfin, dans le paragraphe 3.2.4, nous introduisons une pondération des arcs de manière à pouvoir également compter chaque occurrence d'un motif de causalité observé.

3.2.1 Définition formelle d'un DIG-DAG

Une façon naturelle de stocker un ensemble de mots est d'utiliser un automate, où les mots peuvent être lus sur les chemins de ce dernier. Dans ce but, nous proposons une structure de données $\mathcal{D} = (V, E, \lambda, \mathcal{A})$, baptisée DIG-DAG pour *graphe orienté d'intervalles - graphe orienté acyclique*, munie des propriétés suivantes :

(D1) (V, E) est un graphe orienté acyclique avec V comme ensemble de sommets, E comme ensemble d'arcs, et $u_0 \in V$ un unique sommet de degré entrant 0 appelé *racine*.

(D2) λ est une fonction d'étiquetage qui à tout sommet de V associe un symbole de Σ, ou le mot vide ε s'il s'agit de la racine u_0. Par confort d'écriture, il est possible de considérer l'extension de cette fonction d'étiquetage à un chemin du DIG-DAG : $\lambda(u...v) = \lambda(u)...\lambda(v)$.

(D3) $\mathcal{A} \subseteq V$ est l'ensemble des *sommets actifs*. La racine est toujours active. Nous notons $\mathcal{A}_\sigma$ l'ensemble des sommets actifs étiquetés par σ.

(D4) Pour chaque symbole σ, un sommet u possède au plus un successeur v étiqueté par σ. Dans ce cas, on écrit $v = g(u, \sigma)$. L'arc (u, v) est appelé σ-transition.

(D5) L'ensemble des mots formés par tous les chemins depuis la racine est exactement $\mathcal{L}(w)$, et l'ensemble des mots correspondant aux chemins depuis la racine vers n'importe quel sommet actif est exactement $\mathcal{L}^c(w)$.

Définition 5 (DIG-DAG) *Un DIG-DAG pour un préfixe correct w est une structure de données $\mathcal{D} = (V, E, \lambda, \mathcal{A})$ satisfaisant les propriétés (D1) à (D5).*

Cette structure de données est très proche de celle d'un automate fini déterministe. Dans le cas du DIG-DAG, les étiquettes sont toutefois associées aux états plutôt qu'aux transitions, $g(.,.)$ joue un rôle de fonction de transition et u_0 celui d'état initial. La propriété (D4) impliquer que le DIG-DAG est déterministe. Sous cet angle, $\mathcal{L}(w)$ est le langage reconnu quand chaque état est terminal et $\mathcal{L}^c(w)$ est le langage reconnu lorsque seuls les états actifs sont terminaux. Pour un même mot w, plusieurs instances de DIG-DAG peuvent satisfaire les propriétés ci-dessus. Nous cherchons à construire une structure en ligne qui peut-être efficacement mise à jour à la réception d'un nouveau symbole.

3.2.2 Algorithme de stockage naïf et DIG-trie

La solution de stockage des motifs de causalité la plus simple consiste en la construction d'un arbre. Notre construction est l'adaptation de l'arbre préfixe de [Ukkonen, 1995] : la structure est étendue à la réception de chaque symbole, depuis les sommets actifs ($\mathcal{A}$ dans notre cas). Tandis que [Ukkonen, 1995] traite un mot d'entrée, nous traitons un DIG (représenté par son mot correct), qui est une structure plus générale. La relation temporelle ne peut pas être retrouvée à partir de liens suffixes. Contrairement à Ukkonen, nous gérons donc directement l'ensemble des sommets actifs sans passer par des liens suffixes.

Définition 6 (DIG-trie) *Soit $w \in \mathcal{L}$ un préfixe correct. Un DIG-trie associé à w, noté $\mathcal{T}(w)$ est un DIG-DAG dont la structure de graphe est un arbre enraciné.*

À une réindexation près des sommets, cette structure est définie de manière unique par w. La construction en ligne d'un DIG-trie est obtenue en appliquant l'algorithme 1 à chaque symbole de w et en partant du DIG-DAG trivial $(\{u_0\}, \emptyset, \lambda : u_0 \mapsto \varepsilon, \{u_0\})$.

Algorithme 1 : UPDATE_DIG-TRIE(σ, $\mathcal{T}(w)$)

Entrées : $\sigma \in \Sigma \cup \overline{\Sigma}$, $\mathcal{T}(w) = (V, E, \lambda, \mathcal{A})$, le DIG-trie de w.
Output : $\mathcal{T}(w\sigma)$, le DIG-trie de $w\sigma$.

```
si σ ∈ Σ alors
    pour u ∈ A faire
        si u n'a pas de σ-transition et λ(u)σ ∈ C alors
            Créer un nouveau sommet v ∉ V;
            V ← V ∪ {v} ; E ← E ∪ {(u, v)};
            λ(v) ← σ;
            A ← A ∪ {v};
        sinon
            si λ(u)σ ∈ C alors A ← A ∪ {g(u, σ)};
sinon
    A ← A \ A_σ̄ ;                                   // Avec σ̿ = σ
renvoyer (V, E, λ, A).
```

Théorème 3 *Soit $w \in (\Sigma \cup \overline{\Sigma})^*$ et $\sigma \in \Sigma \cup \overline{\Sigma}$. Si $w\sigma$ est un préfixe correct, alors* UPDATE_DIG-TRIE$(\mathcal{T}(w), \sigma) = \mathcal{T}(w\sigma)$.

Preuve. La preuve repose sur le théorème 2. Nous distinguons deux cas, selon si σ appartient à Σ ou bien à $\overline{\Sigma}$.

Le cas le plus simple est celui où $\overline{\sigma} \in \overline{\Sigma}$. En effet, la sous-suite admissible de w reste la même : $\phi(w) = \phi(w\overline{\sigma})$ et l'algorithme laisse V, E et λ inchangés. Ainsi, l'ensemble des chemins étiquetés du DIG-trie $\mathcal{T}(w)$ partant de la racine u_0, caractérisant le langage $\mathcal{L}(w)$, est inchangé avec l'ajout de $\overline{\sigma}$. Quant aux sommets actifs, il suffit de remarquer que $\mathcal{A}_{\overline{\sigma}}$ est l'ensemble des sommets actifs correspondant à un mot prolongeable finissant par $\overline{\sigma}$.

Dans l'autre cas, lorsque $\sigma \in \Sigma$. D'après le théorème 2, un sous-mot admissible de $w\sigma$ qui n'est pas un sous-mot admissible de w appartient à $\mathcal{L}^c(w)\sigma \cup \Sigma^*\mathcal{C}$. Mais, l'ensemble des sommets actifs étant l'ensemble des sommets tels que chaque chemin de la racine vers un de ces sommets corresponde à un sous-mot prolongeable, à chaque fois qu'un sommet u est un sommet actif avec $\lambda(u)\sigma \in \mathcal{C}$, il est nécessaire d'activer ou de créer un sommet $g(u, \sigma)$, ce qui est exactement ce que fait l'algorithme. □

Exemple 9 *Le DIG-trie de la figure 3.5 est construit à partir du mot $w_{\leq 9} = cb\bar{c}a\bar{b}c\bar{a}db$. Dans la figure 3.5j, les deux sommets étiquetés par b ont un prédécesseur actif et $cb \in \mathcal{C}$, ils*

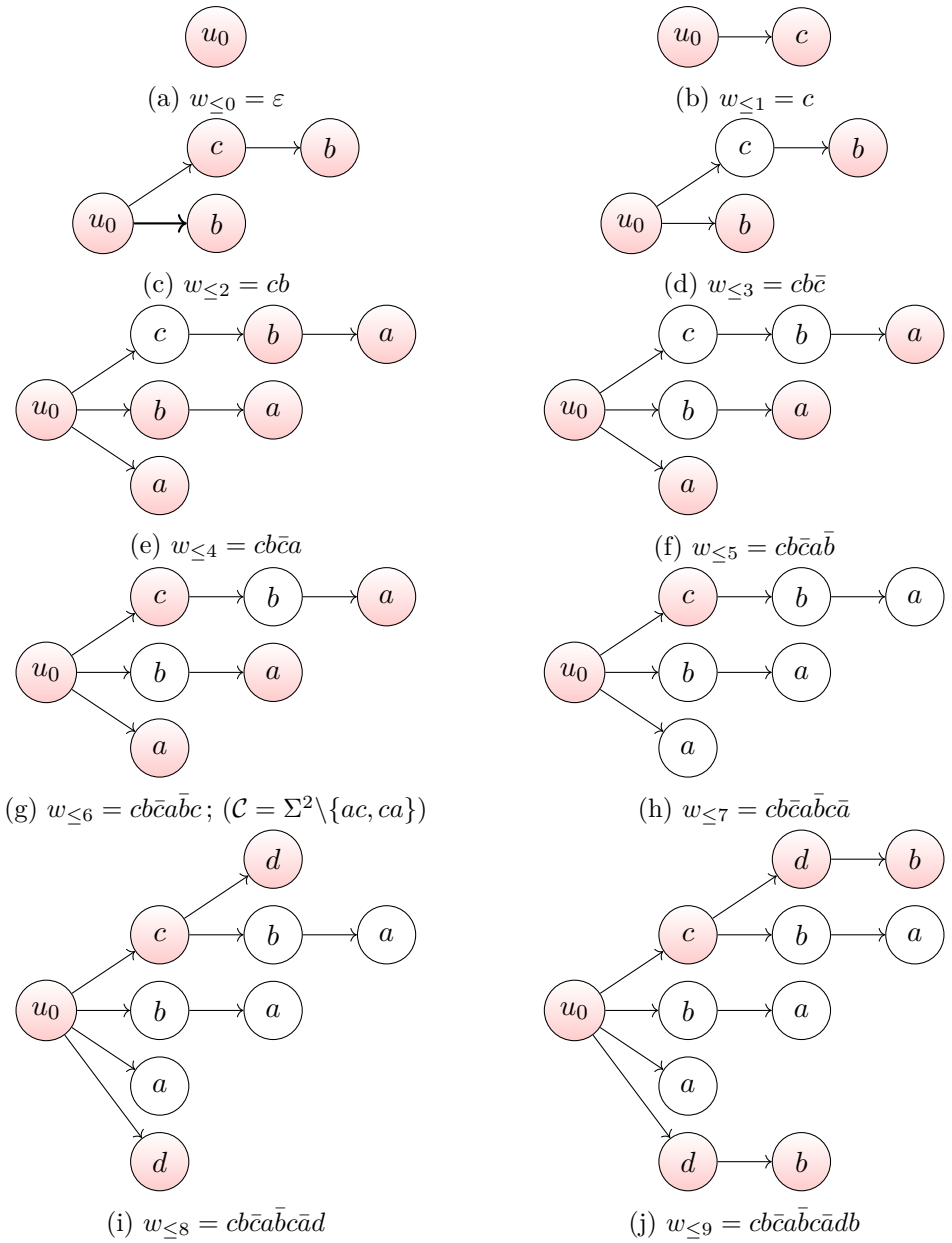


FIGURE 3.5 – Construction du DIG-trie de $w_{\leq 9} = cb\bar{c}a\bar{b}c\bar{a}db$. Les sommets rouges correspondent aux sommets actifs, les blancs sont les sommets inactifs.

sont ainsi réactivés. Comme $db \in \mathcal{C}$, deux nouveaux sommets étiquetés par b sont créés à la suite des deux sommets étiquetés par d.

3.2.3 Algorithme de stockage optimisé et DIG-DAG compact

D'après ce qu'il est possible d'observer à travers l'exemple illustré en figure 3.5, la construction du DIG-trie n'est pas réalisable en pratique : le nombre de sommets croît de manière exponentielle avec la taille du log, et la structure devient très redondante. En effet, des sous-graphes identiques apparaissent de nombreuses fois ($d \to b$ ou $b \to a$) au sein du DIG-trie alors qu'ils pourraient être agrégés tout en conservant le même langage reconnu.

Dans ce paragraphe, nous proposons une construction plus efficace qui garde la structure compacte par rapport au DIG-trie. L'idée derrière notre construction est de créer, à chaque étape, aussi peu de nouveaux sommets que possible.

Afin d'être clairs, nous décomposons la procédure de mise à jour en trois étapes, bien qu'il soit possible de tout faire en une seule. L'idée ici est de *fusionner* les sommets redondants. Cependant, la fusion de sommets peut engendrer le stockage de chemins de causalité non observés, par conséquent il est parfois nécessaire de *scinder* un état en deux pour garantir la correction de la structure. L'*extension* et l'*activation*, tirées de l'algorithme 1, demeurent le mécanisme principal de la construction d'un DIG-DAG.

Supposons que $\mathcal{A}$ soit l'ensemble des sommets actifs et que le prochain symbole du mot considéré soit $\sigma \in \Sigma$.

Scission

Soit u un sommet actif tel que $v = g(u, \sigma)$. Nécessairement, $\lambda(u)\sigma \in \mathcal{C}$. Activer directement v peut induire des incohérences dans l'ensemble des mots stockés si v possède également des prédécesseurs inactifs $u' \notin \mathcal{A}$. Pour éviter ça, on scinde le sommet en deux, v et un nouveau sommet v^s, de sorte que les prédécesseurs de v (resp. v^s) soient tous actifs (resp. inactifs). Le sommet v^s a les mêmes successeurs que v. Cette étape est illustrée par la figure 3.6.

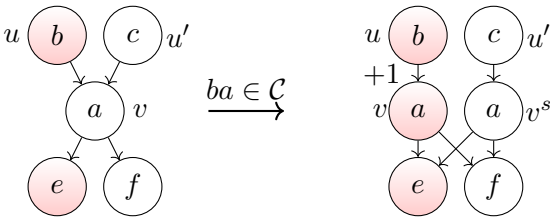


FIGURE 3.6 – Scission : en tant que prédécesseur du sommet étiqueté par b, v doit être activé, mais puisqu'il est aussi prédécesseur du sommet étiqueté par c, il ne doit pas l'être non plus. Un nouveau sommet v^s étiqueté par a est ainsi créé.

Extension et activation

Après l'étape de scission, pour chaque sommet actif $u \in \mathcal{A}$ tels que $\lambda(u)\sigma \in \mathcal{C}$, nous activons $g(u, \sigma)$ s'il existe déjà. Sinon, nous créons un nouveau successeur de u étiqueté par σ. Cette étape est similaire à la mise à jour du DIG-trie.

Fusion

Afin de maintenir la structure compacte, on fusionne toutes les feuilles actives étiquetées par σ en un seul sommet. Le langage des préfixes corrects n'est pas affecté par cette opération.

Toute la procédure de mise à jour du DIG-DAG est donnée par l'algorithme 2.

Exemple 10 *La construction à la volée du DIG-DAG de* $w_{\leq 9} = cb\bar{c}a\bar{b}c\bar{a}d$ *est décrite dans la figure 3.7. Dans la figure 3.7j, le sommet étiqueté par* b *n'a que des prédécesseurs actifs et* $cb \in \mathcal{C}$*, ainsi, ce sommet est activé. Le sommet avec l'étiquette* d *ne possède pas de* b*-transition, un nouveau sommet est donc créé.*

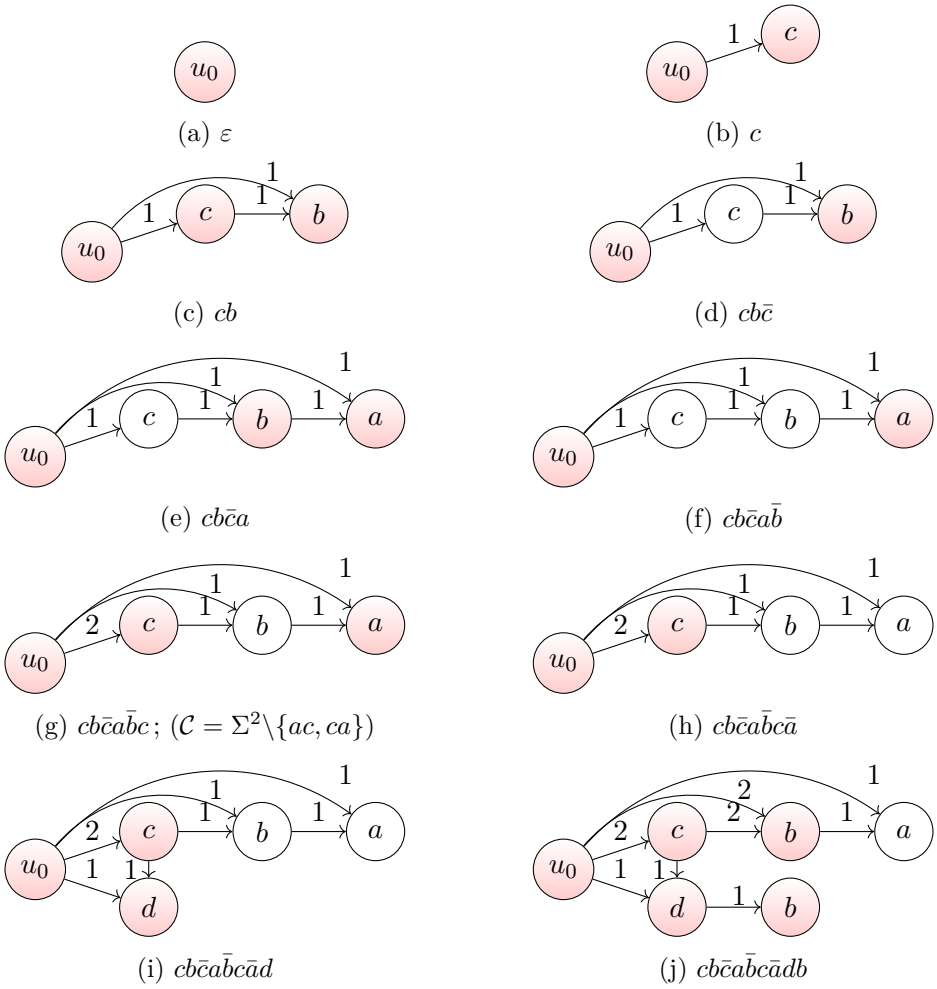
Montrons à présent que l'algorithme 2 construit le DIG-DAG de $w\sigma$ à partir de celui traitant w et de la réception du symbole σ.

Théorème 4 *Soit* $w \in (\Sigma \cup \overline{\Sigma})^*$ *et* $\sigma \in \Sigma \cup \overline{\Sigma}$*. Si* $w\sigma$ *est un préfixe correct et si* $\mathcal{D}$ *est un DIG-DAG pour* w*, alors* UPDATE_DIG-DAG$(\mathcal{D}, \sigma)$ *est un DIG-DAG pour* $w\sigma$*.*

Algorithme 2 : UPDATE_DIG-DAG $(\mathcal{D}, \sigma)$

Entrées : $\sigma \in \Sigma \cup \overline{\Sigma}$, $\mathcal{D} = (V, E, \lambda, \mathcal{A})$ un DIG-DAG pour w.
Output : un DIG-DAG pour $w\sigma$.

si $\sigma \in \Sigma$ **alors**
  `// Scission`
  pour $u \in \mathcal{A}$ *tel que* $g(u, \sigma) = v$ *soit défini* **faire**
    si $\exists u' \notin \mathcal{A}$ *avec* $g(u', \sigma) = v$ **alors**
      Créer un nouveau sommet $v^s \notin V$; $V \leftarrow V \cup \{v^s\}$;
      pour $u' \notin \mathcal{A}$ *t.q.* $g(u', \sigma) = v$ **faire**
        $E \leftarrow E \cup \{(u', v^s)\} \setminus \{(u', v)\}$
      pour $(v, u') \in E$ **faire** $E \leftarrow E \cup (v^s, u')$;
  `// Extension, activation and fusion`
  Créer un nouveau sommet v_ℓ ; $\lambda(v_\ell) \leftarrow \sigma$; $\mathcal{A} \leftarrow \mathcal{A} \cup \{v_\ell\}$;
  pour $u \in \mathcal{A}$ *et* $\lambda(u)\sigma \in \mathcal{C}$ **faire**
    si $g(u, \sigma) = v$ *est défini et n'est pas une feuille* **alors**
      $\mathcal{A} \leftarrow \mathcal{A} \cup \{v\}$
    sinon $E \leftarrow E \cup \{(u, v_\ell)\} \setminus \{(u, g(u, \sigma))\}$;
sinon $\mathcal{A} \leftarrow \mathcal{A} \setminus \mathcal{A}_{\overline{\sigma}}$;
Supprimer les sommets isolés si besoin;
renvoyer $(V, E, \lambda, \mathcal{A})$.



(a) ε (b) c (c) cb (d) $cb\bar{c}$ (e) $cb\bar{c}a$ (f) $cb\bar{c}a\bar{b}$ (g) $cb\bar{c}a\bar{b}c$; $(\mathcal{C} = \Sigma^2 \backslash \{ac, ca\})$ (h) $cb\bar{c}a\bar{b}c\bar{a}$ (i) $cb\bar{c}a\bar{b}c\bar{a}d$ (j) $cb\bar{c}a\bar{b}c\bar{a}db$

FIGURE 3.7 – Construction et pondération du DIG-DAG de $cb\bar{c}a\bar{b}c\bar{a}d$. La pondération des arcs est expliquée dans le paragraphe 3.2.4.

Preuve. Le théorème 4 est prouvé en maintenant, pendant la mise à jour, la correspondance des sommets du DIG-trie vers ceux du DIG-DAG, qui préserve les sommets actifs.

Notons $\Gamma^{\mathcal{D}}_{u,v}$ l'ensemble des chemins du DIG-DAG $\mathcal{D}$ commençant par le sommet u et terminant par le sommet v ($\mathcal{D}$ est un DAG, plusieurs chemins peuvent exister entre deux sommets).

Soit $w \in \mathcal{L}$ un préfixe correct. Supposons qu'il existe une projection π des sommets de $\mathcal{T}(w)$ vers les sommets de $\mathcal{D}(w)$ telle que

- la racine du DIG-trie est projetée sur celle du DIG-DAG :

$$\pi(u_0^{\mathcal{T}}) = u_0^{\mathcal{D}};$$

- le chemin étiqueté de la racine du DIG-trie vers un sommet $u \in V^{\mathcal{T}}$ correspond à un chemin de la racine du DIG-DAG vers son projeté $\pi(u)$:

$$\forall u \in V^{\mathcal{T}}, \lambda(u_0^{\mathcal{T}}...u) \in \{\lambda(\gamma); \gamma \in \Gamma^{\mathcal{D}}_{u_0^{\mathcal{D}},\pi(u)}\};$$

- si un sommet du DIG-trie est actif alors son projeté l'est aussi :

$$u \in \mathcal{A}^{\mathcal{T}} \Rightarrow \pi(u) \in \mathcal{A}^{\mathcal{D}};$$

- si un sommet du DIG-trie est inactif alors son projeté l'est aussi :

$$u \notin \mathcal{A}^{\mathcal{T}} \Rightarrow \pi(u) \notin \mathcal{A}^{\mathcal{D}}.$$

Soit $\sigma \in \Sigma$ un symbole nouvellement reçu. Nous souhaitons mettre à jour la projection π en une projection π' de $\mathcal{T}(w\sigma)$ vers $\mathcal{D}(w\sigma)$.

À la réception de σ, pour tout sommet actif $u^{\mathcal{D}} \in \mathcal{A}^{\mathcal{D}}$, le sommet $v^{\mathcal{D}}$ σ-successeur de $u^{\mathcal{D}}$ est soit déjà existant, soit une feuille nouvellement créée, soit la copie d'un sommet existant (après une scission), soit la fusion de plusieurs sommets. Alors, il suffit de projeter les σ-successeurs de tout sommet du DIG-trie projeté sur $u^{\mathcal{D}}$ par π sur le sommet $v^{\mathcal{D}}$:

$$\forall u^{\mathcal{T}} \in \pi^{-1}(u^{\mathcal{D}}),\ \pi'(g(u^{\mathcal{T}}, \sigma)) = v^{\mathcal{D}};$$

Tous les sommets $g(u^{\mathcal{T}}, \sigma)$ correspondent aux sommets actifs de $\mathcal{T}(w\sigma)$ projetés sur les sommets actifs $v^{\mathcal{D}}$ de $\mathcal{D}(w\sigma)$.

Pour tous les autres sommets u, c'est-à-dire les sommets qui ne sont pas des σ-successeurs de sommets actifs de $\mathcal{D}(w)$, la projection est inchangée : $\pi'(u) = \pi(u)$. Les sommets considérés correspondent à la racine de $\mathcal{T}(w\sigma)$ projetée sur la racine de $\mathcal{D}(w\sigma)$ d'une part, et aux sommets inactifs de $\mathcal{T}(w\sigma)$ projetés sur ceux de $\mathcal{D}(w\sigma)$ d'autre part.

Ce qui permet de prouver le théorème 4 par récurrence. □

3.2.4 Pondération des arcs

Si le DIG-DAG stocke l'ensemble des motifs de causalité potentielle observés dans le log, il est souhaitable de distinguer les motifs les plus fréquents de ceux les plus rares pour caractériser la pertinence de chacun. Pour commencer, nous devons compter, au fur et à mesure de la construction du DIG-DAG, le nombre de fois que chaque motif a été vu. Par exemple, dans la figure 3.3, le contexte *cba* apparaît deux fois tandis que le contexte *dba* n'apparaît qu'une seule fois.

Soient $u \in \mathcal{D}$ un sommet et $\sigma \in \Sigma$ un symbole. Nous appelons *contexte de* u l'ensemble $\Gamma^{\mathcal{D}}_{u_0,u}$ des chemins étiquetés de la racine jusqu'à u, et le *contexte de* $u\sigma$ l'ensemble $\Gamma^{\mathcal{D}}_{u_0,u}\sigma$. Notons $n_{u,\sigma}$ le nombre de fois que le contexte $\Gamma^{\mathcal{D}}_{u_0,u}\sigma$ apparaît dans le DIG, c'est-à-dire, le nombre de fois qu'un chemin de $\Gamma^{\mathcal{D}}_{u_0,u}\sigma$ a été observé dans le DIG.

Intuitivement, les propriétés suivantes doivent être vérifiées :

- Le poids d'un arc nouvellement créé vaut 1 puisqu'il correspond à un motif vu pour la première fois ;
- Le poids d'un arc est incrémenté de 1 dès que le sommet cible de cet arc est réactivé.

Alors que ces propriétés sont évidentes à maintenir dans le cas du DIG-trie (chaque sommet sauf u_0 est la cible d'un unique arc et chaque motif correspond à un unique mot), cette tâche devient un peu plus ardue pour d'autres DIG-DAG. Heureusement, dans l'algorithme 2, les fusions ne concernent que les feuilles et la mise à jour des poids ne consiste qu'en l'incrémentation du poids des arcs dont la cible a été activée lors de l'étape d'activation. La scission et la fusion ne mettent en jeu que des copies des arcs, et les poids sont eux aussi copiés. Si les DIG-DAG obtenus grâce à l'algorithme 2 sont compatibles avec une telle pondération, ce n'est pas le cas de n'importe quel DIG-DAG. La section suivante (3.3.1) en donne un exemple.

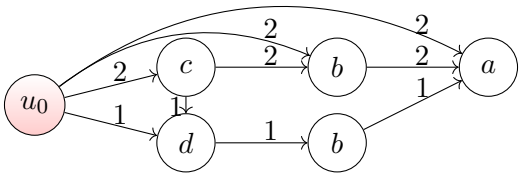


FIGURE 3.8 – Le DIG-DAG pondéré de $w = cb\bar{c}a\bar{b}c\bar{a}db\bar{c}d\bar{a}a\bar{b}$.

Exemple 11 *Les figures 3.6 et 3.7 montrent comment la pondération des arcs du DIG-DAG est maintenue au fil de la réception des symboles. Les poids finaux de notre exemple jouet sont donnés par la figure 3.8.*

3.3 Variantes

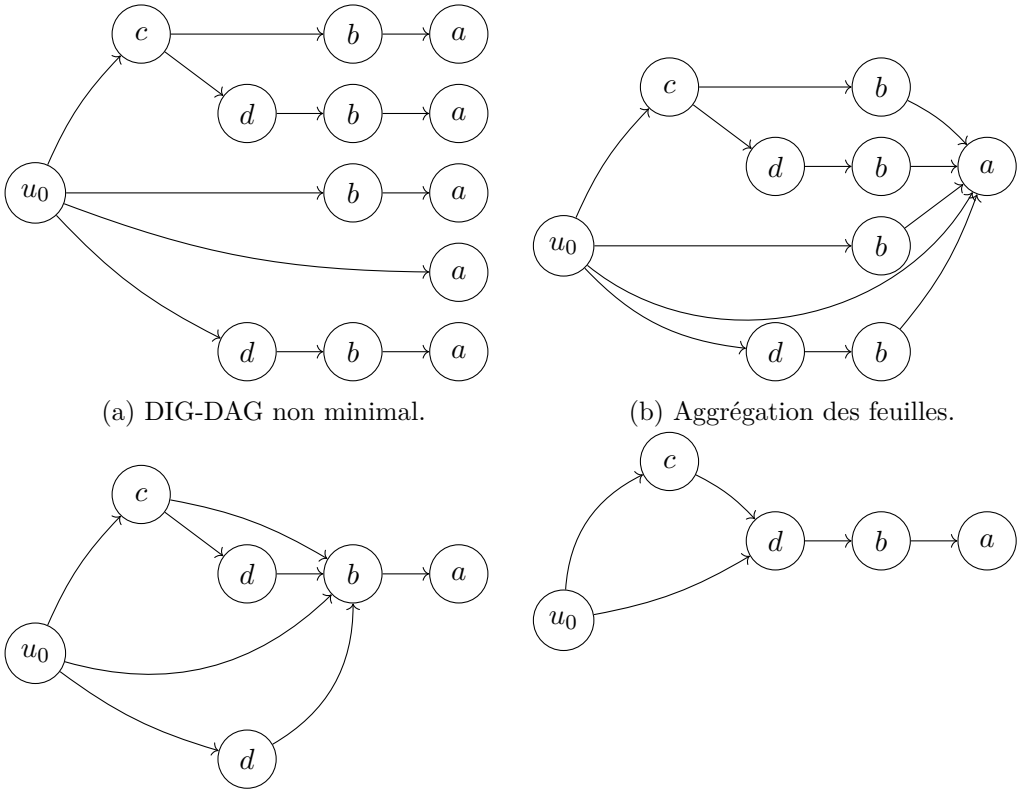
Dans cette section, nous abordons quelques extensions et variantes au DIG-DAG présenté ci-dessus. Plusieurs DIG-DAG différents peuvent être construits à partir d'un même DIG, le paragraphe 3.3.1 discute ainsi de la pertinence du DIG-DAG minimal. Puis, le paragraphe 3.3.2 discute de la possibilité de stocker dans les sommets du DIG-DAG non pas des symboles mais des mots afin de rendre plus compacte la structure.

3.3.1 Minimalité

Le DIG-DAG construit avec l'algorithme 2 n'est pas minimal en général. Par exemple, dans la figure 3.8, les deux sommets étiquetés par b peuvent être fusionnés. Maintenir une structure minimale requiert des opérations qui, contrairement à l'algorithme 2, peuvent mettre en jeu des ancêtres distants des sommets actifs. En particulier, [Revuz, 1992] permet de minimiser un automate fini déterministe acyclique et est parfaitement adapté au DIG-DAG. Il peut être aisément adapté à notre contexte avec quelques modifications mineures (à cause de l'étiquetage des sommets au lieu de celui des arcs).

Cet algorithme de minimisation s'applique à des automates finis déterministes munis d'une structure acyclique, ce qui correspond très bien au DIG-DAG. Cette minimisation peut être réalisée en temps linéaire, grâce à une exploration du graphe des feuilles vers la racine. Une fonction de hauteur des sommets est définie. Cet algorithme parcourt l'automate

considéré en partant de ses feuilles et en remontant le long de la structure à minimiser (approche *bottom up*). Les états sont considérés successivement par “courbes de niveau” (l'ensemble des sommets de même hauteur), de sorte à ne pas modifier le langage reconnu. Un exemple est donné en figure 3.9. La position horizontale des sommets est déterminée en fonction de leur hauteur. La hauteur des feuilles vaut 0 ; celle des autres sommets est calculée récursivement comme le maximum de la hauteur des enfants incrémenté de 1. Par exemple, la hauteur de la racine vaut ici 4. Les sommets peuvent alors progressivement être fusionnés en prenant garde de ne pas altérer le langage reconnu.



(a) DIG-DAG non minimal.

(b) Aggrégation des feuilles.

(c) Des feuilles vers les racines.

(d) DIG-DAG minimisé.

FIGURE 3.9 – Minimisation d'un DIG-DAG étape par étape avec l'algorithme de Revuz [Revuz, 1992].

Par ailleurs, même s'il existe des algorithmes efficaces de minimisation dans ce cas, leur temps d'exécution reste significatif. Cela impose de geler la structure le temps de la minimisation, ce qui ne permet pas un traitement immédiat des alarmes apparaissant dans ce laps de temps. Nous verrons par ailleurs dans la section 5.2.1 que le DIG-DAG

construit avec l'algorithme 2 conserve une taille proche du DIG-DAG minimal. Il s'agit d'un bon compromis car cela permet d'obtenir une structure assez compacte sans minimisations supplémentaires.

De plus, notons que la procédure de minimisation empêche la conservation de la pondération. En effet, l'étape de minimisation *a posteriori* fusionne des arcs aux poids potentiellement différents. Ce point et celui mentionné dans le paragraphe précédent sont les raisons pour lesquelles nous ne nous restreignons pas à des DIG-DAG minimaux.

3.3.2 Étiquetage des transitions par des mots

À l'image d'Ukkonen, qui considère à la fois des tries et des arbres, on pourrait essayer d'adapter la construction du DIG-trie pour construire des DIG-trees. En l'occurrence, Ukkonen compresse les parties du graphe qui ne sont que des chaînes : les étiquettes sont alors des sous-mots. Pour cela, les sous-mots sont repérés par une paire d'indices. Les indices du mot w sont facilement généralisés par des identifiants de sommets dans le DIG. Cependant, puisqu'il est possible d'avoir plusieurs chemins entre deux sommets, les facteurs ne peuvent plus être représentés par des paires d'indices. Mais on pourrait garder l'idée d'agréger les chemins composés de sommets de degrés sortants égaux à 1 en un seul sommet du DIG-tree voire même du DIG-DAG. Il n'est toutefois pas clair qu'une telle modification soit intéressante. La topologie de cette structure est certes plus petite, mais chaque sommet stocke désormais une paire d'indices (2×4 bytes) au lieu d'un seul caractère (typiquement 1 byte). Par simplicité, nous avons décidé d'écarter cette éventuelle optimisation.

3.4 Conclusion

Dans ce chapitre, nous avons présenté une structure appelée DIG-DAG qui stocke l'ensemble des motifs d'alarmes construits à partir d'un log, conformément à une relation de causalité *spatio-temporelle*. Il est également possible de considérer les connaissances *a priori* d'un expert afin d'écarter des relations de causalité potentielle non pertinentes, et de compter le nombre d'occurrences de chaque motif. L'obtention d'une structure recensant et comptant l'ensemble des motifs observés peut faire office de base à plusieurs applications comme la visualisation et la compréhension des motifs de causalité mis en jeu dans le log étudié,

l'exploration de l'ensemble de ces motifs à la recherche de motifs pertinents pour l'explication de pannes, ou encore la prédiction de pannes puisque sa construction est faite en ligne.

Toutefois, étant donné la taille d'un log d'alarmes, stocker toutes les chaînes de causalité potentielle observées prend de la place : le DIG-DAG résultant est volumineux et difficilement exploitable en l'état. En particulier, de nombreuses chaînes stockées ne sont pas pertinentes à l'analyse et il est nécessaire de pouvoir isoler efficacement les chaînes pertinentes des autres. Ainsi, dans le chapitre suivant, nous permettons à l'expert d'extraire des motifs spécifiques stockés dans un DIG-DAG grâce à un système de requêtes.

Chapitre 4

Exploitation du DIG-DAG

Dans le chapitre 3, nous avons présenté une structure capable de stocker l'ensemble des motifs de causalité potentielle observés au sein d'un log, appelée DIG-DAG. Cette structure fournit ainsi la représentation graphique d'un log, enrichi d'une notion de causalité. À partir d'un log, on peut construire un DIG-DAG en ligne, ce qui est une condition nécessaire pour la prédiction de pannes. Cependant, il est difficile de l'utiliser directement pour prédire des pannes. En effet, cette tâche requiert d'inscrire le DIG-DAG dans un cadre probabiliste rigoureux, ce que nous n'avons pas encore réussi à faire. Dans un premier temps, nous cherchons donc plutôt à exploiter le DIG-DAG pour l'analyse de causes racines. Toutefois, le DIG-DAG peut être très volumineux selon le log donné en entrée et donc encore difficilement exploitable par un expert. Parmi tous les motifs observés et stockés, seul un petit nombre est généralement pertinent pour l'analyse de logs. Dans ce chapitre, nous proposons un ensemble d'outils pour extraire et visualiser un sous-ensemble de motifs de causalité potentielle, définis selon les critères de l'utilisateur. Ces outils ont d'abord été pensés pour être utilisés par un opérateur humain de façon hors ligne, et donc plutôt dans un contexte d'analyse de causes racines.

Ainsi, dans la section 4.1, après avoir caractérisé ce qu'est un sous-ensemble de motifs de causalité grâce à la notion de sous-DIG-DAG, nous explicitons ce qu'est une requête sur un DIG-DAG, puis décrivons comment obtenir le sous-DIG-DAG correspondant. Nous proposons ensuite un ensemble d'opérations visant à simplifier un sous-DIG-DAG pour mieux le visualiser dans la section 4.2. Les outils présentés dans ce chapitre sont tirés de travaux publiés dans un article publié à MLN [Salaün et al., 2019a].

4.1 Extraction de motifs pertinents

Dans cette section, nous présentons une solution permettant d'extraire d'un DIG-DAG ou d'un sous-DIG-DAG les chaînes de causalité potentielle conformes à une recherche spécifiée par l'utilisateur.

Définition 7 (sous-DIG-DAG) *Soient $\mathcal{D} = (V, E, \lambda, \mathcal{A})$ un DIG-DAG, et (V', E') un sous-graphe de (V, E). Le quadruplet $\mathcal{D}' = (V', E', \lambda_{|V'}, \mathcal{A} \cap V')$ est un sous-DIG-DAG de $\mathcal{D}$.*

On peut remarquer qu'un sous-DIG-DAG n'est pas nécessairement un DIG-DAG : il pourrait très bien avoir plusieurs racines ou être non connexe.

4.1.1 Requêtes

Nous cherchons à présent à formaliser un système de requête conforme au modèle introduit par la définition 7 et pratique pour l'utilisateur.

Définition 8 (Requête et sous-DIG-DAG correspondant) *Une requête est définie par un quintuplet $\mathcal{Q} = (\mathcal{D}, S, T, V_i, E_i)$, où $\mathcal{D} = (V, E, \lambda, \mathcal{A})$ est un DIG-DAG, $S, T, V_i \subseteq V$ et $E_i \subseteq E$. Le résultat de cette requête est le plus grand sous-DIG-DAG $(V', E', \lambda', \mathcal{A}')$ de $\mathcal{D}$ tel que :*

- $V' \subseteq V_i, E' \subseteq E_i$;
- *les sommets de degré entrant* 0 *sont tous dans* S ;
- *les sommets de degré sortant* 0 *sont tous dans* T.

Le résultat de la requête est noté $\mathcal{D}'(\mathcal{Q})$.

Intuitivement, le sous-DIG-DAG de $\mathcal{D}$ issu d'une requête $\mathcal{Q} = (\mathcal{D}, S, T, V_i, E_i)$ est le sous-graphe de $\mathcal{D}$ dont les chemins maximaux partent tous de S et terminent dans T. Ces chemins ne passent que par les sommets de V_i et les arcs de E_i.

4.1.2 Requêtes régulières

La définition de requête est très large et, dans ce paragraphe, nous nous restreignons aux requêtes paramétrées par un automate fini et des propriétés locales sur les sommets et sur les arcs. Intuitivement, le rôle de l'automate fini est de stocker des motifs satisfaisant certaines relations entre les sommets, tandis que les propriétés locales permettent de sélectionner arcs et sommets. Ces propriétés ne dépendent pas seulement des symboles portés par les sommets (elles auraient pu sinon être inclues dans un automate), mais reposent sur tout type de métadonnées rattachées aux sommets. Par exemple, il peut être utile d'extraire des sommets récemment actifs ou de ne conserver que des arcs pertinents (pertinence définie à partir de la pondération décrite en section 3.2.4).

Définition 9 (Requêtes régulières) *Soient $\mathcal{M}$ un automate fini, ainsi que $P_v : V \to \{0,1\}$ et $P_e : E \to \{0,1\}$ deux propriétés (respectivement sur les sommets et sur les arcs). On définit la requête régulière $\mathcal{R}(S,T,\mathcal{M},P_v,P_e) = (S,T,V_i,E_i)$, où :*

- *pour tout $u \in V_i$, u satisfait P_v ;*
- *pour tout $(u,v) \in E_i$, (u,v) satisfait P_e ;*
- *$\mathcal{D}'(\mathcal{R}(S,T,\mathcal{M},P_v,P_e))$ contient tous les chemins étiquetés de $\mathcal{D}'((\mathcal{D},S,T,P_v,P_e))$ reconnus par $\mathcal{M}$. Par conséquent, S_i et T_i sont respectivement définis par l'ensemble des sommets et des arcs appartenant à ces chemins ;*
- *$\mathcal{D}'(\mathcal{R}(S,T,\mathcal{M},P_v,P_e))$ est le plus petit sous-DIG-DAG de $\mathcal{D}$ (au sens de l'inclusion) satisfaisant ces propriétés.*

L'algorithme 3 calcule le sous-DIG-DAG correspondant à une requête régulière donnée. Nous supposons que nous connaissons un ordre topologique sur les sommets. Pour cela, il est possible d'utiliser un algorithme classique (cf. [Cormen et al., 2009] (section 22.4) par exemple).

L'ordre topologique peut aussi être calculé en ligne au moment de l'indexation des sommets lors de la construction du DIG-DAG. En effet, pendant la mise à jour du DIG-DAG (cf. algorithme 2), les nouveaux sommets qui ne sont pas issus d'une scission sont toujours des feuilles : incrémenter leur indexe suffit à entretenir un ordre. Le cas de la fusion est réglé en conservant le maximum des deux indices. Concernant les scissions, il suffit de décider d'un sous-ordre arbitraire entre les copies. Il n'est alors plus utile de recalculer cet ordre topologique par la suite.

Algorithme 3 : REGULAR_QUERY$(\mathcal{D}, \mathcal{M}, S, T, P_v, P_e)$

Entrées : $\mathcal{D} = (V, E, \lambda, \mathcal{A}), \mathcal{M} = (Q, \Sigma, \delta, I, F), S, T, P_v, P_e$
Output : un sous-DIG-DAG $\mathcal{D}'$
`// Phase 1: parcours dans le sens direct`
pour chaque $u \in V$ **faire** $Q_1(u) \leftarrow \emptyset$;
pour chaque $u \in V$ *(dans un ordre topologique)* **faire**
  si $u \in S \cap P_v$ **alors** $Q_1(u) \leftarrow Q_1(u) \cup I$;
  pour chaque $v \in V \cap P_v$ *tel que* $(u, v) \in E \cap P_e$ **faire**
    $Q_1(v) \leftarrow Q_1(v) \cup \{q' \in Q \mid \exists q \in Q_1(u),\ \delta(q, \lambda(v)) = q'\}$
`// Phase 2: parcours en sens inverse`
$V' \leftarrow \emptyset$; $E' \leftarrow \emptyset$;
pour chaque $u \in V$ **faire** $Q_2(u) \leftarrow \emptyset$;
pour chaque $u \in V$ *(dans l'ordre topologique inverse)* **faire**
  si $u \in T \cap P_v$ **alors** $Q_2(u) \leftarrow Q_2(u) \cup (Q_1(u) \cap F)$;
  si $Q_2(u) \neq \emptyset$ **alors**
    $V' \leftarrow V' \cup \{u\}$
    pour chaque v *tel que* $(v, u) \in E \cap P_e$ **faire**
      si $\{q \in Q_1(v) \mid \exists q' \in Q_2(u),\ \delta(q, \lambda(u)) = q'\} \neq \emptyset$ **alors**
        $E' \leftarrow E' \cup \{(v, u)\}$;
        $Q_2(v) \leftarrow Q_2(v) \cup \{q \in Q_1(v) \mid \exists q' \in Q_2(u),\ \delta(q, \lambda(u)) = q'\}$
renvoyer $\mathcal{D}' = (V', E', \lambda_{|V'}, \mathcal{A} \cap V')$

L'algorithme 3 est réalisé en deux étapes : la première identifie, par un parcours dans le sens direct, tous les chemins possibles partant de S, dont les sommets et arcs vérifient respectivement P_v et P_e, et dont les étiquettes forment des préfixes de mots reconnus par l'automate. Pour cela, un ensemble $Q_1(u)$ est attribué à chaque sommet $u \in V$, contenant tous les états de l'automate qui peuvent être atteints depuis un sommet de S.

La seconde étape consiste en un parcours inverse de $\mathcal{D}$ qui identifie les sommets et les arcs qui appartiennent au sous-DIG-DAG désiré. Pour chaque sommet u, $Q_2(u)$ est le sous-ensemble d'états de $Q_1(u)$ tel qu'il y ait un chemin de u à un sommet de T qui corresponde à un chemin d'un état de $Q_2(u)$ vers un état terminal de l'automate. Les sommets et les arcs impliqués dans ces chemins constituent le sous-DIG-DAG souhaité.

Plus formellement, soit $\mathcal{M} = (Q, \Sigma, \delta, I, F)$ un automate fini, où Σ représente son alphabet, Q l'ensemble de ses états, I ses états initiaux, F ses états terminaux et δ sa fonction de transition. Nous représentons le DIG-DAG comme un automate : l'étiquette d'une transition (u, v) du DIG-DAG correspond à l'étiquette $\lambda(v)$ du sommet cible. Notons que $\mathcal{M}$ ne

peut pas être utilisé pour conditionner la pertinence d'un chemin selon l'étiquette portée par le sommet de départ (ce qui peut être pris en charge par P_v).

Nous montrons ensuite que le résultat renvoyé par l'algorithme 3 est le sous-DIG-DAG correspondant à la requête régulière définie dans la définition 9. Exposons tout d'abord quelques propriétés de $Q_2(u)$.

Lemme 2 *Avec les notations ci-dessus,*

- *pour tout $u \in V'$ et $q \in Q_2(u)$, soit il existe $v \in V'$ et $q' \in Q_2(v)$ tels que $(u, v) \in E'$ et $q' = \delta(q, \lambda(v))$, soit $u \in T$ et $q \in F$;*
- *pour tout $v \in V'$ et $q' \in Q_2(v)$, soit il existe $u \in V'$ et $q \in Q_2(u)$ tels que $(u, v) \in E'$ et $q' = \delta(q, \lambda(v))$, soit $v \in S$ et $q' \in I$;*
- *pour tout $v \in V'$, pour tout $q' \in Q_2(v)$, il existe un chemin d'un sommet $u \in S$ vers v correspondant à un chemin dans $\mathcal{M}$ d'un état initial $q \in I$ vers q' ; u et v (resp. q et q') peuvent être confondus ;*
- *pour tout $v \in V'$, pour tout $q' \in Q_2(v)$, il existe un chemin de v vers un sommet $u \in T$ correspondant à un chemin dans $\mathcal{M}$ de q' vers un état terminal $q \in F$; v et u (resp. q' et q) peuvent être confondus.*

Preuve. Commençons par prouver le premier point. Soit $u \in V'$ un sommet du sous-DIG-DAG renvoyé par l'algorithme 3. D'après la ligne 10, $Q_2(u) \neq \emptyset$ puisque u a été retenu dans le sous-DIG-DAG. Soit $q \in Q_2(u)$. Cet état a été introduit soit à la ligne 15, soit à la ligne 9. Dans le premier cas, l'état q est rajouté après que le test de la ligne 13 soit passé : u est le prédécesseur d'un sommet v dans le sous-DIG-DAG (voir ligne 14) tel qu'il existe un état $q' \in Q_2(v)$ où $\delta(q, \lambda(v))$ (ligne 15). Ou alors, cet état a été introduit au moment de la ligne 9, ce qui implique $u \in T$ et $q \in F$.

Pour prouver le second point du lemme, considérons un sommet $v \in V'$. D'après la ligne 10, $Q_2(v) \neq \emptyset$. D'après les lignes 9 et 15, $Q_2(v)$ est un sous-ensemble de $Q_1(v)$. Ainsi, tout état $q' \in Q_2(v)$ est d'abord introduit dans $Q_1(v)$ par les lignes 3 et 5. S'il a été introduit dans 3, alors $v \in S$ et $q \in I$. Sinon, il a été introduit au moment de 5 : il existe un sommet $u \in V \cap P_v$ et un état $q \in Q_1(u)$ tels que $\delta(q, \lambda(v)) = q'$. Puisque $q' \in Q_2(v)$, alors q appartient aussi à $Q_2(u)$ (ligne 15) et u est un sommet du sous-DIG-DAG (ligne 11). Par ailleurs, l'arc (u, v) est un arc du sous-DIG-DAG d'après la ligne 14.

Finalement, les deux derniers points du lemme sont obtenus récursivement grâce aux deux premiers. □

Théorème 5 *Considérons la requête régulière $\mathcal{R}(S, T, \mathcal{M}, P_v, P_e)$. Soit $\mathcal{D}'$ le sous-DIG-DAG renvoyé par l'algorithme 3. Nous avons alors $\mathcal{D}' = \mathcal{D}'(\mathcal{R}(S, T, \mathcal{M}, P_v, P_e))$.*

Preuve. Nous devons vérifier les quatre propriétés de la définition 9. Les deux premières viennent directement, puisque tous les sommets et arcs ajoutés à V' et E' vérifient respectivement P_v et P_e (lignes 9 et 12).

Vérifions à présent la troisième propriété : soit $u_1, ..., u_f$ un chemin dans $\mathcal{D}$ étiqueté par un mot accepté par $\mathcal{M}$, tel que $u_1 \in S$ et $u_f \in T$. Soit $q_1, ..., q_f$ une suite d'états visités de $\mathcal{M}$ acceptant ce mot. Pour tout i, par construction, $q_i \in Q_1(u_i)$ (ligne 5). Comme $q_f \in F$ et $u_f \in T$, $q_f \in Q_2(u_f)$ (ligne 9), et alors, $q_i \in Q_2(u_i)$ pour tout i (ligne 14) : tous les arcs du chemin sont conservés.

Finalement, nous devons encore vérifier que les chemins extraits sont conformes au langage de $\mathcal{M}$, c'est-à-dire vérifier que tous les arcs du graphe renvoyés par l'algorithme appartiennent à un chemin étiqueté par un mot reconnu par $\mathcal{M}$. C'est une conséquence du lemme 2 : considérons un arc (u, v), il est possible de construire un chemin $u_s \in S \rightsquigarrow u \rightarrow v \rightsquigarrow v_t \in T$ correspondant à $q_i \in I \cap Q_2(u_s) \rightsquigarrow q \in Q_2(u) \rightarrow q' \in Q_2(v) \rightsquigarrow q_f \in F \cap Q_2(v_t)$ en appliquant les points 3, et 4 du lemme 2. □

Ce théorème prouve ainsi que le sous-DIG-DAG obtenu est bel et bien le plus petit contenant l'intersection de $\mathcal{D}$ et $\mathcal{M}$. Ainsi, l'algorithme 3 renvoie le sous-DIG-DAG de la requête régulière correspondante. Par ailleurs, on peut remarquer que le langage reconnu par le sous-DIG-DAG n'est pas le langage reconnu par $\mathcal{M}$

Exemple 12 *Dans cet exemple, nous extrayons du DIG-DAG représenté en figure 4.1a les motifs satisfaisant l'expression régulière $\Sigma^* d \Sigma^*$, dont l'automate $\mathcal{M}$ correspondant est représenté en figure 4.1b. Nous choisissons par ailleurs $S = \{u_0\}$ et $T = V$; les propriétés P_v et P_e acceptent tout. Les sommets du DIG-DAG ont été numérotés selon un ordre topologique. L'expression régulière est représentée par l'automate illustré par la figure 4.1b. Les étapes de l'algorithme 3 appliqué à cet exemple sont illustrées dans la figure 4.2. La première phase va de 4.2a à 4.2e, tandis que la seconde phase correspond aux figures de 4.2f*

à 4.2k. Les ensembles $Q_1(.)$ sont indiqués à côté de leurs sommets respectifs pour la phase 1, et $Q_2(.)$ pour la phase 2. Les arcs et sommets ajoutés à V' et E' figurent en gras, et ceux non sélectionnés en pointillés.

Le sous-DIG-DAG en gras de la figure 4.2k est bien celui conforme à la requête régulière $\mathcal{R}(S, T, \mathcal{M}, P_v, P_e)$.

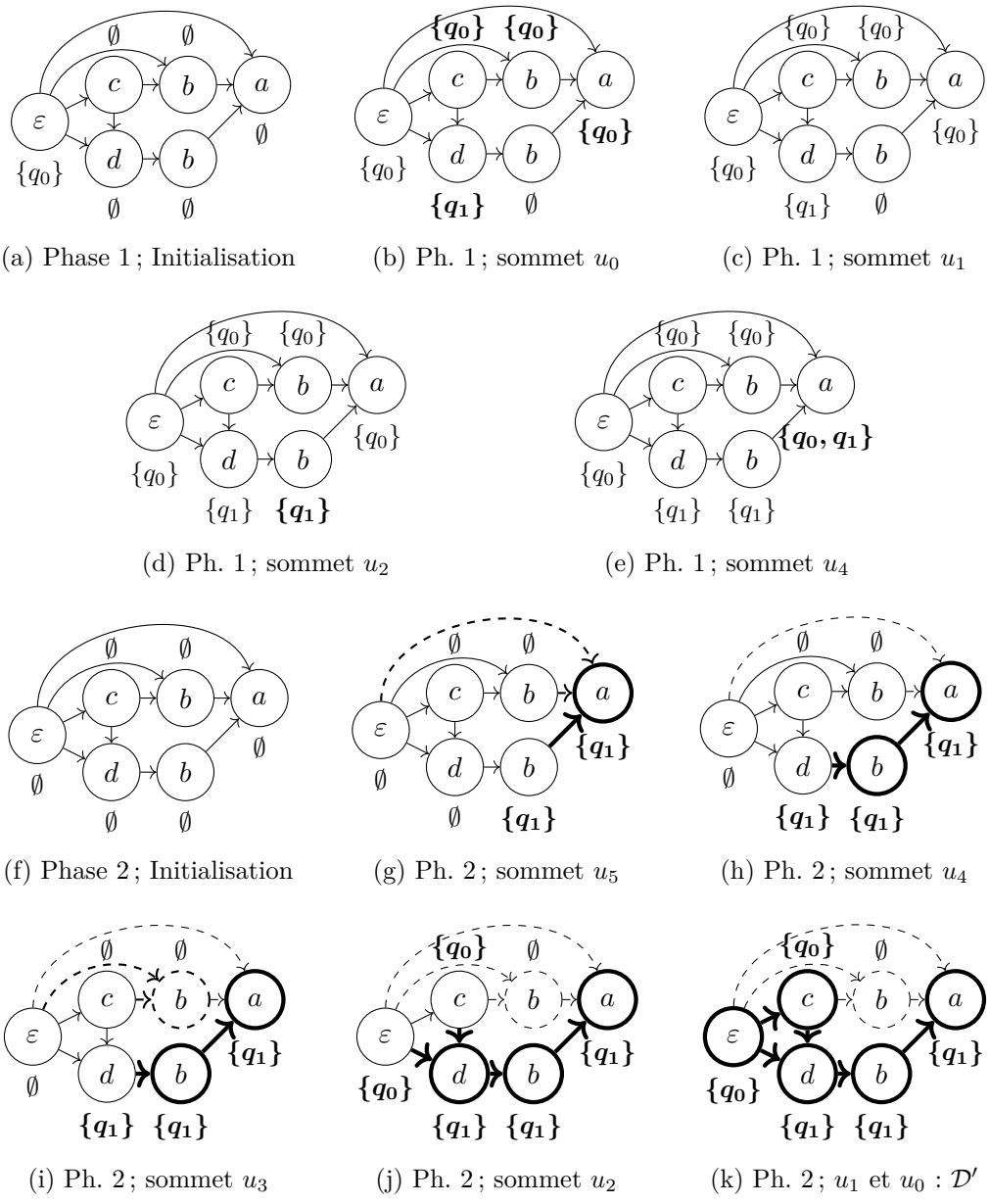
(a) DIG-DAG. (b) Automate.

FIGURE 4.1 – DIG-DAG et automate soumis à l'algorithme 3, utilisé dans l'exemple 12.

4.2 Simplification d'un sous-DIG-DAG

Bien que le système de requête permette de spécifier quelles parties du DIG-DAG extraire, le sous-DIG-DAG obtenu peut être grand selon la requête présentée par l'expert. En effet, le DIG-DAG peut être dense et les motifs recherchés peuvent apparaître à divers endroits dans la structure. Afin d'améliorer la lisibilité du résultat, cette section présente trois manières de simplifier le sous-DIG-DAG extrait.

Notons que ces simplifications ne peuvent être considérées comme des opérations d'extraction de sous-DIG-DAG, puisqu'elles peuvent altérer la structure originale de DIG-DAG, en fusionnant des sommets, ou en n'étant pas compatibles avec la notion de poids développée dans le paragraphe 3.2.4. Ce n'est pas un inconvénient trop important puisque nous introduisons ces opérations pour améliorer la représentation graphique.



(a) Phase 1 ; Initialisation (b) Ph. 1 ; sommet u_0 (c) Ph. 1 ; sommet u_1

(d) Ph. 1 ; sommet u_2 (e) Ph. 1 ; sommet u_4

(f) Phase 2 ; Initialisation (g) Ph. 2 ; sommet u_5 (h) Ph. 2 ; sommet u_4

(i) Ph. 2 ; sommet u_3 (j) Ph. 2 ; sommet u_2 (k) Ph. 2 ; u_1 et u_0 : $\mathcal{D}'$

FIGURE 4.2 – Algorithme 3 appliqué au DIG-DAG et à l'automate de la figure 4.1.

4.2.1 Réduction transitive

Le but de la réduction transitive est ici de retirer les arcs qui semblent redondants à la lecture. Introduite dans [Aho et al., 1972], cette opération supprime tous les arcs (u, v) tel qu'il existe un autre chemin de u vers v. La réduction transitive retire ainsi certains chemins du sous-DIG-DAG mais garde les plus longs.

L'algorithme 5 permet de réaliser la réduction transitive d'un graphe dirigé acyclique $\mathcal{G} = (V, E)$. Pour cela, il détermine l'ensemble des arcs qui sont des « raccourcis » de chemins plus longs (voir définition 10) et les retire de $\mathcal{G}$.

Définition 10 (Raccourci) *Un arc $(u, v) \in E$ d'un graphe acyclique est qualifié de raccourci s'il existe un autre chemin de u vers v.*

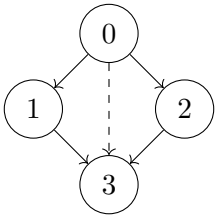


FIGURE 4.3 – Réduction transitive sur un exemple jouet. Les arcs pleins comme en pointillés sont ceux du graphe : les arcs en pointillés montrent des raccourcis à supprimer, tandis que les arcs pleins sont à conserver.

Exemple 13 *Dans la figure 4.3, l'arc $(0, 3)$ en pointillé est un raccourci pour le chemin $0 \to 1 \to 3$.*

Pour tout arc $e \in E$, nous notons $E^-(e)$ (resp. $E^+(e)$) ses arcs entrants (resp. arcs sortants). Pour tout sommet $u \in V$, nous notons $V^-(u)$ (resp. $V^+(u)$) ses sommets entrants (resp. sommets sortants). Pour pallier toute ambiguïté, un sommet entrant (resp. sortant) est la source d'un arc entrant (resp. la cible d'un arc sortant).

Pour réaliser la réduction transitive de $\mathcal{G}$, l'algorithme est réalisé en deux étapes. Dans un premier temps, pour chaque sommet $v \in V$, nous déterminons l'ensemble de ses ancêtres

dénotés $A(v)$, c'est-à-dire des sommets u tels qu'il existe un chemin de u vers v. En particulier, connaître les ancêtres d'un sommet v permet de déterminer pour chaque arc e sortant de v, l'ensemble des sommets u tels qu'il existe un chemin partant de u et passant par e. En effet, si deux arcs ciblent un même sommet, l'un des deux peut être le raccourci d'un chemin plus long se terminant par le second arc et cela se vérifie ensuite en consultant les ancêtres de leurs sources respectives (voir exemple 15). L'algorithme 4 permet d'établir la liste $A(v)$ de chaque sommet u.

Algorithme 4 : ANCETRES($\mathcal{G}$)

```
   // Pour chaque sommet v ∈ V, trouver tous les sommets u tels qu'il
      existe un chemin de u vers v. pa
   Entrées : un graphe dirigé acyclique G = (V, E)
   Output : l'application A : V → P(V) qui associe à chaque sommet l'ensemble de
            ses ancêtres
 1 Soit S une pile contenant les racines de G ;
 2 pour chaque u ∈ V faire
 3 |   A(u) ← ∅ ;
 4 |   ν(u) ← |V⁻(u)| ;
 5 tant que S ≠ ∅ faire
 6 |   v ← dépiler(S) ;
 7 |   A(v) ← {v} ;
 8 |   pour chaque u ∈ V⁻(v) faire
 9 |   |   A(v) ← A(v) ∪ A(u) ;
10 |   pour chaque u ∈ V⁺(v) faire
11 |   |   ν(u) ← ν(u) − 1 ;
12 |   |   si ν(u) = 0 alors S ← S ∪ {u};
13 renvoyer A
```

Afin de lister les ancêtres de chaque sommet, nous parcourons les sommets du graphe à partir de ses racines. Les sommets sont passés en revue petit à petit à l'aide d'une pile S qui contient les sommets à considérer dans un premier temps (lignes $1, 5, 6$ et 12). Les ancêtres d'un sommet sont déterminés à partir de ceux de ses parents (ligne 9). Pour que cela puisse être fait, il faut que tous les ancêtres de tous ses parents aient été complètement identifiés. Une fois que tous les sommets entrants d'un sommet ont été considérés, les ancêtres de ce sommet ne peuvent plus changer et on peut alors l'ajouter à la pile des sommets à traiter. Pour maintenir l'avancement du traitement de chaque sommet u, on compte dans $\nu(u)$ les arcs entrants qu'il reste à étudier (ligne 11). Naturellement, ceci est rendu possible par le fait que les graphes étudiés sont acycliques et que parcourir le graphe en suivant les arcs ne

permet pas de traiter deux fois un même sommet.

Exemple 14 *Les ancêtres des sommets du graphe illustré en figure 4.3 sont* $A(0) = \{0\}$, $A(1) = \{0, 1\}$, $A(2) = \{0, 2\}$ *et* $A(3) = \{0, 1, 2, 3\}$.

Lemme 3 *Chaque sommet* $u \in V$ *est considéré au plus une fois dans la pile* S.

Preuve. Soit $u \in V$. D'après la ligne 11, la quantité $\nu(u)$ est décroissante (il s'agit du seul endroit où elle est mise à jour). Par ailleurs, d'après les lignes 1 et 12, u ne peut être ajouté à la pile S que si $\nu(u) = 0$.

Ainsi, dans le cas où u aurait déjà appartenu une fois à S, alors $\nu(u) \leq 0$. Rajouter ce sommet une seconde fois à S requiert de passer par la ligne 11, ce qui implique $\nu(u) < 0$. Le test de la ligne 12 est alors nécessairement négatif et u ne peut appartenir de nouveau à S. □

Le lemme 3 permet de conclure que l'algorithme 4 se termine bel et bien. En effet, un sommet est dépilé à chaque appel de la boucle de la ligne 5. Puisque la régénération de la pile S est limitée par le lemme 3, S finit par se vider et terminer la boucle.

Dans un second temps, nous confrontons chaque arc aux autres arcs qui ciblent le même sommet. En effet, si le sommet source d'un arc est inclus dans les ancêtres de la source d'un autre arc partageant la même cible, alors le premier arc est un raccourci du second puisqu'il évite de passer par certains sommets. Il peut donc être supprimé. Cette étape est décrite par l'algorithme 5.

Une fois l'ensemble des prédécesseurs de la source de chaque arc déterminé, l'ensemble E_r des arcs à supprimer est établi. Pour chaque sommet v, nous en considérons les arcs entrants. Tout d'abord, ces arcs sont triés selon le nombre décroissant de prédécesseurs (ligne 5). Notamment, l'arc qui a le plus de prédécesseurs ne saurait être un raccourci. Ensuite, du second arc ayant le plus de prédécesseurs à celui en ayant le moins, nous veillons à ce que chaque arc ne soit pas un raccourci. Pour cela, il suffit de noter les ancêtres observés A_p depuis les arcs qui ont été attestés ne pas être des raccourcis (lignes 7 et 12) et vérifier que la source de l'arc en question n'est pas dedans (ligne 9).

Par construction, un DIG-DAG ne possède pas d'arcs multiples, ce cas n'est donc pas traité. Dans le cas d'un graphe acyclique en possédant, il peut être nécessaire de rajouter une

Algorithme 5 : REDUCTION_TRANSITIVE($\mathcal{D}$)

```
   // Identifier les raccourcis et les retirer.
   Entrées : un graphe dirigé acyclique G = (V, E)
   Output : un sous-graphe du graphe d'entrée G'
1  A ← ANCETRES(G) ;
2  E_r ← ∅ ;
3  pour chaque v ∈ V faire
4      si V⁻(v) ≠ ∅ alors
5          Trier E⁻(v) par nombre décroissant d'ancêtres du sommet source ;
           // Prévoir une règle pour départager les cas d'égalité induits
              par d'éventuels arcs multiples.
6          (u, v) ← E⁻(v)[0] ;
7          A_p ← A(u) ;
8          pour chaque (u, v) ∈ E⁻(v)[1 :] faire
9              si u ∈ A_p alors
10                 E_r ← E_r ∪ {(u, v)} ;
11             sinon
12                 A_p ← A_p ∪ A(u) ;
13 renvoyer G' = (V, E\E_r)
```

règle pour différencier ces arcs et de conserver le plus pertinent (ou de les fusionner en un seul). Cela se fait au moment de définir le tri des arcs entrants par nombre de prédécesseurs de leurs sources.

Exemple 15 *Dans le graphe de la figure 4.3, les ancêtres de* 0 *(source de* $(0, 3)$*) sont inclus dans ceux de* 1 *(source de* $(1, 3)$*) : l'arc* $(0, 3)$ *est donc un raccourci car tous les ancêtres de* 1 *qui ne figurent pas parmi les ancêtres de* 0 *sont ceux (en l'occurrence* 1*) évités par le raccourci lors du chemin* $0 \to 1 \to 3$*.*

Les arcs $(1, 3)$ *et* $(2, 3)$ *ont des ancêtres communs mais sans pour autant marquer d'inclusion : ils ne sauraient être des raccourcis l'un de l'autre.*

Question complexité, le lemme 3 nous garantit que les boucles des lignes 8 et 10 sont appelées au plus $|V|$ fois. Puisque la boucle 8 (resp. 10) considère les arcs entrants (resp. sortants) de chaque sommet, lui-même considéré au plus une fois d'après le lemme 3, les lignes $9, 11$ et 12 sont appelées au plus $|E|$ fois. Ce qui donne la complexité de l'algorithme 4 (et donc de la ligne 1 de l'algorithme 5).

De même, pour l'algorithme 5, les lignes 9 à 12 sont appelées au plus $|E|$ fois. Enfin, la ligne 5 est appelée $|V|$ fois et a une complexité en $O(|E|\log(|E|))$. La complexité totale de l'algorithme 5 est donc $O(|E| + |V||E|\log(|E|)) = O(|V||E|\log(|E|))$.

4.2.2 Minimisation

Comme indiqué dans la section 3.3.1, le DIG-DAG obtenu à partir de l'algorithme 2 n'est pas minimal et peut être minimisé avec des outils tels que l'algorithme de Revuz [Revuz, 1992]. Pour rappel, cet algorithme commence par fusionner les feuilles portant le même symbole. Puis, le DIG-DAG est minimisé petit à petit en remontant vers la racine. Un exemple de minimisation est montré en figure 3.9.

Or, d'après les sections 3.2.1 et 4.1.2, un DIG-DAG peut être vu comme un automate déterministe acyclique. L'algorithme de Revuz peut ainsi être utilisé sur un sous-DIG-DAG puisqu'il s'agit d'un sous-graphe du DIG-DAG qui peut donc être considéré également comme un automate déterministe pourvu qu'il ne soit doté que d'un seul sommet source.

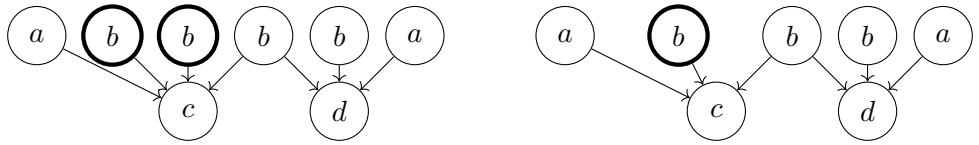
Cela permet d'exhiber des structures minimales décrivant exactement les mêmes motifs de causalité que le sous-DIG-DAG de départ. Rappelons que la minimisation ne préserve pas la pondération de la structure. En effet, certains sommets ou arcs peuvent être fusionnés, ne garantissant plus une pondération correcte de ces derniers.

La minimisation n'est pas souhaitable au cours de la construction du DIG-DAG mais peut s'avérer pratique quand les résultats renvoyés par les requêtes sont grands et difficiles à lire. Ce n'est pas gênant de faire une minimisation dans ce contexte car cette opération peut se faire sur une copie du sous-DIG-DAG extrait, sans influer sur le fonctionnement du DIG-DAG.

4.2.3 Simplification des racines d'un sous-DIG-DAG

Il peut y avoir de nombreux sommets portant la même étiquette, et surtout correspondants à la même occurrence d'un événement. Lors d'une requête, l'ensemble S peut avoir été décrit par une certaine propriété et plusieurs racines du sous-DIG-DAG considéré peuvent alors porter la même étiquette. Nous avons observé en pratique que beaucoup d'entre elles avaient les mêmes successeurs. La simplification des racines d'un sous-DIG-DAG consiste à

réunir, tant que possible, deux racines différentes $u_s, u'_s \in S$ si et seulement si $\lambda(u_s) = \lambda(u'_s)$. Un exemple jouet est donné en figure 4.4.



(a) Sous-DIG-DAG avec des racines redondantes.

(b) Sous-DIG-DAG après simplification de ses racines.

FIGURE 4.4 – Simplification des racines d'un sous-DIG-DAG jouet. Les sommets en gras sont ceux qui peuvent être agrégés (figure 4.4a) et qui résultent de cette agrégation (figure 4.4b).

Les sommets de degré entrant 0 sont simplement passés en revue. Une table des sources déjà visitées est maintenue : pour chaque source, on vérifie qu'elle n'a pas déjà été observée. Si tel est le cas, on l'ajoute à la table sinon on l'ajoute à la liste des sources à retirer. La complexité de cette opération est donc en $O(|V|^2)$. Là encore, cette opération n'est en général pas compatible avec la pondération développée en paragraphe 3.2.4 : deux racines identiques du point de vue des étiquettes et de leurs successeurs peuvent arborer des arcs sortants pondérés différemment.

4.3 Conclusion

Dans ce chapitre, nous avons proposé un système de requêtes pour extraire des explications synthétiques et pertinentes à partir de structure de graphes de causalité. L'outil introduit est suffisamment flexible pour permettre à un expert de retrouver à l'intérieur du log des chaînes de causalité conformes à ses recherches et en particulier, conformes à une expression rationnelle. Ce système de requête a d'abord été pensé pour une utilisation hors ligne par un opérateur humain dans un contexte d'analyse de causes racines. Ainsi, afin de faciliter davantage la lecture des motifs extraits, nous avons également proposé une série d'outils de simplification d'un sous-DIG-DAG. Ces outils, bien qu'ils en altèrent la structure, en clarifient la visualisation.

Chapitre 5

Applications expérimentales

Par construction, le DIG-DAG est une structure stockant toutes les chaînes de causalité observées dans un log. Le fait qu'il soit possible de le construire en ligne laisse envisageable son utilisation pour la prédiction de pannes. Toutefois, nous nous sommes d'abord concentrés sur son utilisation en analyse de causes racines. En particulier, nous avons mis au point un système de requêtes pour extraire des sous-DIG-DAG à partir des spécifications de l'utilisateur. Dans ce chapitre, nous développons les aspects pratiques liés au DIG-DAG et, plus particulièrement, à son utilisation en analyse de causes racines.

En section 5.1, nous commençons par expliciter la façon dont les outils de construction et d'exploitation du DIG-DAG présentés dans les chapitres 3 et 4 peuvent être associés à travers une méthodologie pour l'analyse de causes racines. Cette méthodologie est réalisée de bout en bout, c'est-à-dire de l'ingestion du log brut à l'extraction de sous-DIG-DAG succints et interprétables. La section 5.2 présente ensuite des données expérimentales concernant le passage à l'échelle de la structure et la capacité du DIG-DAG à stocker des motifs de causalité pertinents. Dans 5.3, nous concluons par l'application de notre méthodologie de bout en bout pour l'analyse de causes racines à un log réel. Cela nous permet de discuter quantitativement des performances de notre solution.

En particulier, l'ensemble des outils présentés ont été implémentés en Python 3. Outre l'acquisition de données expérimentales, cela nous a notamment permis de collaborer avec certaines *business units* de Nokia. Par ailleurs, une démo illustrant cette méthodologie a été présentée au *5G Smart Campus Event* organisé par Nokia Bell Labs puis plus formellement

aux ateliers d'INFOCOM 2020 [Salaün et al., 2020].

5.1 Méthodologie pour l'analyse de bout en bout d'un log d'alarmes

Cette section décrit une approche utilisant l'ensemble des outils introduits par les sections 3.2 et 4.1 pour l'analyse de bout en bout dans un contexte de recherche de causes racines. Dans ce contexte d'analyse de cause racine, la panne a déjà eu lieu. Par conséquent, nous pouvons supposer que le log est donné et qu'il n'est pas nécessaire de le traiter en ligne (bien que certaines étapes puissent être réalisées à la volée). Dans les paragraphes 5.1.1 à 5.1.3, nous expliquons l'approche générale : le paragraphe 5.1.1 aborde les différents points à prendre en compte lors du chargement du log, la construction du DIG-DAG à partir du log ainsi chargé est traitée dans le paragraphe 5.1.2, le paragraphe 5.1.3 explique alors comment extraire les sous-DIG-DAG appropriés à l'analyse de causes racines grâce à notre outil de requête. Enfin, le paragraphe 5.1.4 présente quant à lui quelques conseils pour aborder des logs trop gros pour que l'analyse passe à l'échelle.

5.1.1 Chargement du log

La première étape consiste à transformer le log brut en une suite d'événements. Cela inclut notamment de déterminer l'alphabet, de fixer la période d'activité de chaque événement ainsi que la fenêtre d'observation sur laquelle on souhaite étudier le log.

Définition de l'alphabet. Le choix de l'alphabet (formalisé en section 3.1.1) est décidé à partir des caractéristiques désirées pour l'analyse. En pratique, les caractéristiques les plus pertinentes sont :

- le *nom de l'alarme* : il s'agit en général d'un texte court ou d'un numéro ;
- l'élément émetteur (par abus de langage la *machine*), représenté par son identifiant. Cela peut être une machine, le port d'un switch, la cellule d'un réseau sans fil, etc. ;
- la *sévérité* de l'alarme, représentée par un nombre ou une couleur, qui croit ou s'assombrit avec la sévérité.

Redéfinition des dates de début et de fin. Selon que le log a été prétraité ou non, que les événements soient dotés d'une date de fin ou non, la période d'activité d'un événement doit être attentivement définie.

Dans le cas le plus simple, un événement à une date de début et une date de fin, ce qui définit respectivement la date de début et de fin de la période d'activité.

Toutefois, dans de nombreuses applications, les événements ne sont pas actifs sur un intervalle de temps mais sont déclenchés à un instant donné. Nous appelons ceux-ci des événements *ponctuels*. Dans ce cas, la relation de causalité potentielle temporelle n'est plus définie. Une façon de résoudre ce problème consiste à définir artificiellement la date manquante. Celle-ci peut être choisie de plusieurs manières différentes.

- une *durée fixe pour chaque type d'alarme*. On associe une durée τ_σ à chaque symbole σ. Alors, un événement ponctuel (σ, s) est transformé en un événement segmenté $(\sigma, [s, s + \tau_\sigma])$. Si un autre événement $e' = (\sigma, t)$ avec $s < t < s + \tau_\sigma$ apparaît, alors e' est supprimé et l'événement segmenté remplacé par $(\sigma, [s, t + \tau_\sigma])$, et ainsi de suite. Les valeurs de τ_σ peuvent être par exemple choisies selon le comportement moyen des événements dans le log ;
- un *nombre de dépendances potentielles pour chaque événement*. Fixons k_σ ce nombre pour chaque symbole σ. Quand l'événement (σ, s) est reçu, nous autorisons une dépendance avec les k_σ prochains événements (σ_i, t_i) tels que $\sigma\sigma_i \in \mathcal{C}$ et que t_i fasse partie des k_σ plus petits instants parmi les instants t_j vérifiant $t_j \geq s$.
- Une *combinaison des deux points précédents*. Par exemple, il est possible d'établir un maximum de k_σ dépendances pendant un période τ_σ.

Par ailleurs, on peut réaliser en ligne la conversion d'alarmes ponctuelles vers des alarmes segmentées. Dans les cas décrits ci-dessus, nous pouvons associer une horloge (pour la durée τ_σ) et un compteur (pour le nombre de dépendances k_σ) à chaque sommet. Lorsqu'un sommet est (ré)activé, son horloge (resp. compteur) est (ré)initialisée à τ_σ (resp. k_σ), et cette horloge (resp. compteur) est décrémentée au fil du temps (resp. des arrivées d'événements). Lorsque l'un des deux atteint 0, le symbole $\overline{\sigma}$ est alors émis, désactivant tous les sommets étiquetés par σ. À chaque fois qu'un sommet est désactivé, son horloge et son compteur sont réinitialisés à 0.

Il ainsi possible de traiter des logs sous d'autres formats que celui initialement présenté

dans la section 3.1.1 tout en gardant la représentation du log en mot, utile au traitement en ligne de ce dernier.

Dans certains logs d'alarmes, les temps d'émissions ont lieu à des instants précis. C'est ainsi le cas des KPIs qui sont émis périodiquement (toutes les cinq minutes par exemple comme c'est le cas par défaut de l'outil *Cacti* [section 1.1.1]). Dans ce cas, afin d'assurer la causalité temporelle, un bon choix serait de définir une période d'activité un peu plus longue que la périodicité des mesures.

S'il n'y a pas de telle périodicité, alors l'intervalle de temps peut être défini en étudiant le taux d'arrivée moyen des événements. Ajouter cet intervalle de temps à la date d'émission de l'événement permet ainsi d'obtenir une date de fin.

5.1.2 Construction du DIG-DAG

Une fois le log proprement défini, l'intégration de connaissances *a priori* dans l'ensemble des causalités spatiales potentielles $\mathcal{C}$ permet de filtrer en amont les causalités non pertinentes et ainsi de réduire la taille du DIG-DAG. Voici quelques exemples :

- si aucune information n'est disponible, $\mathcal{C}$ est simplement Σ^2 ;
- si la position géographique des équipements du réseau est connue (antennes et cellules dans un réseau sans fil par exemple), alors $\sigma\sigma' \in \mathcal{C}$ si et seulement si les symboles σ et σ' caractérisent des machines qui ne sont pas éloignées de plus d'un rayon donné ;
- si certains comportements logiques sont connus, comme par exemple un certain type d'élément ou d'application ne peuvent communiquer qu'avec un autre type de machines, il est possible de construire $\mathcal{C}$ à partir de ces communications possibles.

Le DIG-DAG peut alors être construit conformément à $\mathcal{C}$.

En pratique, notons que des informations additionnelles peuvent être attachées à la structure, telles que les poids sur les arcs, définis en paragraphe 3.2.4, ou la dernière date d'activation de chaque sommet.

5.1.3 Requêtes sur le DIG-DAG

Une fois le DIG-DAG construit, il contient tous les motifs spatio-temporels observés au sein du log et il est alors possible d'en extraire des sous-DIG-DAG au travers de notre système de requêtes. Dans le cas d'une requête régulière, les paramètres sont $S, T, \mathcal{M}, P_v, P_e$ (cf. paragraphe 4.1.2). Les paragraphes suivants présentent quelques exemples.

Filtrage des arcs

Cette opération est réalisée en définissant la propriété P_e selon les besoins de l'utilisateur. Dans son analyse, il peut ainsi ne retenir que certains types de liens de causalité et écarter les autres.

Par exemple, supposons que l'on veuille extraire des parties du DIG-DAG telles que les arcs indiquent une forte corrélation entre les sommets, c'est-à-dire extraire les motifs les plus vraisemblables permettant d'expliquer une panne. Nous définissons alors un autre système de pondérations à partir des poids des arcs définis en 3.2.4. Soit u un sommet du DIG-DAG et $\sigma \in \Sigma$ tel que $g(u, \sigma)$ existe. Définissons le ratio $n_{u,\sigma}/n_{u_0,\sigma}$ (le poids $n_{u_0,\sigma}$ est le nombre total d'occurrences de σ dans w). Sans définition rigoureuse de la probabilité $\mathbb{P}$, nous pouvons donner l'interprétation

$$r_{u,\sigma} = \frac{n_{u,\sigma}}{n_{u_0,\sigma}} \simeq \frac{\mathbb{P}(u, \sigma)}{\mathbb{P}(u_0, \sigma)} = \mathbb{P}(u|\sigma).$$

Il s'agit du ratio entre le nombre de fois qu'un motif finissant par l'arc (u, v) a été observé et le nombre d'occurrences du symbole σ dans le log. Ce ratio représente la probabilité que, étant donné l'observation de σ, le contexte $\Gamma^{\mathcal{D}}_{u_0,u}$ soit apparu avant. Si ce ratio est proche de 1, alors σ est positivement corrélé au contexte de u. Si ce ratio est au contraire petit, alors σ et u sont négativement corrélés ou indépendants.

La propriété P_e va ainsi sélectionner les arcs dont le ratio est supérieur à un certain seuil ρ.

Filtrage des sommets

Ce filtrage est fait en définissant la propriété P_v. Par exemple, si on souhaite écarter les alarmes de sévérité basse afin de pouvoir se concentrer sur des alarmes plus importantes, il suffit alors de sélectionner un sous-ensemble de Σ défini d'après ce critère basé sur la sévérité. Si on souhaite plutôt se concentrer sur les événements les plus récents, P_v peut alors être défini pour ne considérer que les sommets qui ont été récemment actifs. De même, il est possible de concentrer son analyse uniquement sur un groupe de machines suspectes.

Choix des sources

Définissons S, les sources possibles des chemins de causalité. Par défaut, on peut choisir $S = \{u_0\}$ pour garder tous les motifs. Au contraire, on pourrait décider de se concentrer sur d'autres types d'événements, comme les événements les plus récents.

Choix des cibles

Pour la définition de T, on pourrait vouloir se concentrer sur des événements critiques. Ces événements peuvent être situés autour d'un pic d'alarmes (révélé lors d'une analyse *a priori*) ou en fin de log. Ainsi, un bon choix est de se concentrer sur les alarmes actives à ces moments-là et qui présentent une sévérité importante.

Requêtes plus spécifiques

Pour des requêtes plus spécifiques, un automate $\mathcal{M}$ peut être défini. Par exemple, pour suivre et vérifier la propagation d'un comportement fautif d'une machine m_1 à une autre machine m_2, on peut définir un automate acceptant les mots de la forme $\Sigma_1^*\Sigma_2^*$, où Σ_i est l'ensemble des alarmes émises par la machine m_i, pour $i \in \{1, 2\}$.

Une fois la requête définie, elle est appliquée au DIG-DAG à l'aide de l'algorithme 3. La lisibilité du sous-DIG-DAG obtenu peut être améliorée grâce aux techniques introduites dans le paragraphe 4.2.

5.1.4 Conseils pour les logs de grande taille

Il peut arriver que les logs soient trop gros pour pouvoir construire le DIG-DAG en temps raisonnable. Dans ce paragraphe, nous proposons deux techniques (non exclusives) pour contourner cette difficulté. La première est basée sur la sélection de parties pertinentes du log, la seconde modifie quant à elle l'alphabet Σ pour rendre le log plus compact.

Redéfinition de la fenêtre d'observation

Même si un log est un fichier et que l'analyse peut porter sur le fichier en entier, cela peut être parfois plus pertinent d'arrêter la construction du DIG-DAG avant la fin du log. En effet, tronquer le log de sorte à ne considérer que les événements qui ont mené à la panne, sans prendre en compte les événements qui en sont les conséquences, permet d'obtenir un DIG-DAG moins volumineux sans perte de motif causal pertinent. Par ailleurs, les requêtes sur le DIG-DAG peuvent tirer parti de l'activité des alarmes à la fin de la fenêtre d'observation qui correspond désormais au début de la panne.

La difficulté réside à déterminer à quel moment tronquer le log. Une telle date peut être déterminée à l'aide d'une analyse *a priori* mais il est également possible de la définir à la volée. Intuitivement, un problème peut être détecté lorsque le processus d'émission d'alarmes dévie de son comportement normal. Pour cela, nous nous intéressons au taux d'arrivée des alarmes, ou d'un sous-ensemble d'alarmes.

Plusieurs techniques ont été proposées pour détecter automatiquement des déviations temporelles de comportement. Elles sont toutes basées sur le suivi du taux moyen d'arrivée des messages : on peut notamment citer le modèle de la moyenne mobile [Enders, 2004] ou [Bouillard et al., 2012]. Nous considérons la seconde de ces solutions, capable de suivre précisément les déviations. [Bouillard et al., 2012] repose sur le nombre cumulé d'événements (débuts) en fonction du temps (voir figure 5.1). Le taux d'arrivée des messages est calculé à la volée. À partir de ces taux, un « tube » délimitant une zone de comportement normal, caractérisée à partir du taux d'arrivée attendu plus ou moins une marge arbitraire, est défini. Un événement est alors considéré comme anormal, si le nombre cumulé d'événements se retrouve à l'extérieur du tube : dans ce cas, le taux d'arrivée est mis à jour. Le flux d'anomalies obtenu peut être lui aussi passé au crible de cet outil de détection d'anomalies. Autrement dit, il peut être réutilisé pour une seconde passe plus lissante, et ce répété

autant de fois que nécessaire. Le choix de la marge rend la détection d'anomalie plus ou moins permissive puisque cela revient à choisir le rayon du tube.

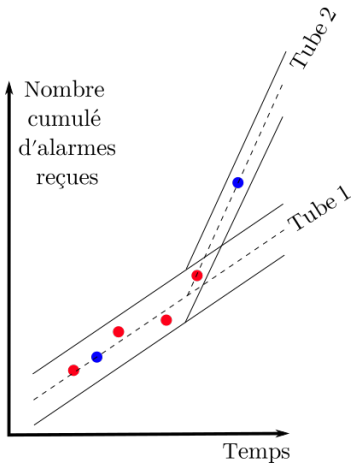


FIGURE 5.1 – Illustration de la détection d'anomalie par [Bouillard et al., 2012]. Les points représentent le nombre cumulé d'alarmes reçues au fil du temps. Dès que les contraintes imposées par le tube actuel (Tube 1), le taux d'arrivée est recalculé et le tube mis à jour (Tube 2). Les points bleus sont ceux qui ont requis la mise à jour du tube ; ils correspondent aux anomalies renvoyées.

Enfin, il est possible de détecter des déviations soudaines grâce à l'inégalité de Bienaymé-Tchebychev [Tchebychev, 1867]. En effet, en calculant la moyenne et la variance des taux d'arrivée, cette inégalité permet de contrôler à tout moment l'écart entre le taux observé τ et celui attendu :

$$\forall \epsilon > 0, \mathbb{P}\left(|\tau - \mathbb{E}[\tau]| \geq \sqrt{\mathrm{Var}(\tau)/\epsilon}\right) \leq \epsilon.$$

En particulier, pour une valeur arbitraire $\epsilon > 0$, $|\tau - \mathbb{E}[\tau]| \geq \sqrt{\mathrm{Var}(\tau)/\epsilon}$ correspond à un écart anormal du comportement attendu. Tous ces outils de détection de déviations permettent d'identifier les alarmes qui ont lieu au moment de ces déviations et de ne considérer qu'elles lors de l'analyse du log.

Réduction de l'alphabet

La taille de l'alphabet a une incidence forte sur la taille du DIG-DAG construit. En effet, le degré sortant de chaque sommet est borné par la taille de l'alphabet. Afin de limiter sa taille, nous identifions et écartons les alarmes non pertinentes, supprimons l'information redondante grâce à une méthode de *clustering*, ou encore isolons les alarmes pertinentes à l'aide de projections sur l'alphabet.

Alarmes non pertinentes. Nous considérons qu'une alarme n'est pas pertinente lorsqu'elle contribue peu à retrouver la cause racine d'un problème. Outre le fait que ce genre d'alarmes n'aident pas à la résolution du problème, leur présence génère l'observation de causalités potentielles inutiles et parasites. Une façon facile de simplifier un log est donc d'ignorer ce genre d'alarmes. Par exemple, des événements dont la sévérité est mineure peuvent être déclarés non pertinents.

Parmi les alarmes non pertinentes, nous pouvons considérer les alarmes fréquentes qui n'apportent pas d'information : elles apparaissent dans tout le log, régulièrement, et quel que soit l'état du système. Dans certains logs qui nous ont été donnés, certains événements de ce genre couvrent ainsi l'intégralité du log : ces événements sont alors considérés par le DIG-DAG comme des sources potentielles de la plupart des événements du log. Les supprimer permet d'améliorer considérablement l'analyse de cause racine en écartant de nombreuses causalités non pertinentes issues de leur forte présence dans le log. Ces alarmes peuvent être déterminées grâce l'observation de logs plus anciens ou en prétraitant le log analysé. Une solution est de mesurer la période totale d'activité de chaque événement et de la comparer à la période couverte par le log.

Grouper les alarmes co-occurrentes. Une autre façon de simplifier le log est de grouper les symboles « co-occurrents », c'est-à-dire les symboles qui s'activent et se désactivent à peu près en même temps. Afin de fusionner les événements correspondants, nous employons une technique classique de clustering ascendante hiérarchique.

Étant donné un ensemble de points, nous souhaitons les regrouper successivement dans des clusters de plus en plus grands. Pour cela, nous avons tout d'abord besoin d'une opération d'union pour fusionner deux clusters, ainsi que d'une mesure de distance afin de ne fusionner que les clusters qui sont proches. L'ensemble des clusters considérés sont stockés

selon une structure de *dendrogramme*, autrement dit, d'arbre binaire étiqueté dont les sommets sont les clusters construits et munis d'une hauteur. La construction de cette structure commence par les feuilles dont les étiquettes identifient les singletons. À chaque étape de la construction, les deux racines les plus proches sont fusionnées en un nouveau parent grâce à l'opération d'union (il s'agit d'une construction gloutonne). La hauteur d'un sommet est définie d'après la distance entre ses fils (les feuilles ont une hauteur nulle). La construction s'arrête lorsqu'il ne reste plus qu'une racine unique qui englobe tous les points.

Dans notre cas, nous souhaitons regrouper les symboles de l'alphabet Σ selon leur ensemble d'intervalles d'activité. Chaque cluster c est caractérisé par deux ensembles I_c et U_c. Pour une feuille, ces ensembles sont tous deux initialisés comme l'union des intervalles d'activité des symboles correspondants. L'opération d'union est alors définie comme suit : pour tous clusters c_1 et c_2, $I_{c_1 \cup c_2} = I_{c_1} \cap I_{c_2}$ et $U_{c_1 \cup c_2} = U_{c_1} \cup U_{c_2}$. Dans un second temps, la relation de distance entre les nœuds est définie par

$$\forall c_1, c_2, d(c_1, c_2) = 1 - \frac{|I_{c_1} \cap I_{c_2}|}{|U_{c_1} \cup U_{c_2}|}.$$

où $|.|$ est la norme L_1. Cette distance est inspirée de la distance de Jaccard entre deux ensembles [Jaccard, 1912]. L'arbre est alors construit petit à petit en appariant les deux racines les plus proches selon cette distance, comme expliqué dans le paragraphe précédent.

Toute coupe transversale du dendrogramme offre une partition de ses feuilles, et donc des symboles de l'alphabet. Plus la hauteur de coupe α est grande, plus les clusters résultants sont grands et moins ils sont nombreux. De plus, parmi les clusters renvoyés, il n'en existe pas deux dont la distance soit plus petite que α. Un nouveau log est généré en considérant chaque cluster c comme un symbole : l'ensemble des intervalles d'activité de ce symbole est alors donné par U_c.

Afin de permettre de telles coupes, il est nécessaire de vérifier l'axiome de reductibilité [Bruynooghe, 1978]. Notons d la distance entre deux clusters et $\cup$ l'opération d'union. Soient trois clusters c, c', c''. Cet axiome s'écrit alors

$$d(c, c') \leq \min\left(d(c, c''), d(c', c'')\right) \Rightarrow \min\left(d(c, c''), d(c', c'')\right) \leq d(c \cup c', c'').$$

Lemme 4 *La méthode de clustering ci-dessus vérifie l'axiome de réductibilité.*

Preuve. Soit c, c', c'' trois clusters. Ils sont définis à l'aide de deux ensembles I_c (resp. $I_{c'}, I_{c''}$) et U_c (resp. $U_{c'}, U_{c''}$). Les opérations d'union et de distance sont celles définies plus haut. Alors,

$$\begin{aligned} d(c \cup c', c'') &= 1 - \frac{|(I_c \cap I_{c'}) \cap I_{c''}|}{|(U_c \cup U_{c'}) \cup U_{c''}|} \\ &\geq 1 - \frac{|I_c \cap I_{c''}|}{|(U_c \cup U_{c'}) \cup U_{c''}|} \\ &\geq 1 - \frac{|I_c \cap I_{c''}|}{|U_c \cup U_{c''}|} \\ &= d(c, c'') \\ &\geq \min\left(d(c, c''), d(c', c'')\right). \end{aligned}$$

□

Une version naïve de cet algorithme aurait pu être envisagée, où seules les unions auraient servi à définir le représentant de chaque sommet et où la distance employée aurait été celle de Jaccard ($\forall c, c', d(c, c') = 1 - \frac{|c \cap c'|}{|c \cup c'|}$). Toutefois, une telle version naïve ne garantit pas l'axiome de réductibilité. En effet, en considèrant l'opération d'union classique et la distance de Jaccard entre deux ensembles, on peut montrer que l'axiome de réductibilité ne tient pas. Pour cela, il suffit d'utiliser un contre-exemple : considérons les intervalles $c = [1, 3], c' = [2, 4]$ et $c'' = [0, 8]$. Nous mesurons alors (les résultats décimaux sont arrondis) :

- $\min\left(d(c, c''), d(c', c'')\right) = d(c, c'') = \frac{3}{4} = 0.75$;
- $d(c, c') = \frac{2}{3} = 0.66$;
- $d(c \cup c', c'') = \frac{5}{8} = 0.63$.

Le membre de gauche de l'axiome de réductibilité est vérifié ($d(c, c') \leq d(c, c'')$) mais pas celui de droite ($d(c, c'') \not\leq d(c \cup c', c'')$).

Projection de l'alphabet et analyse en deux étapes. Une troisième approche consiste à choisir un alphabet moins détaillé qui ne reposerait pas sur toutes les caractéristiques normalement utilisées pour décrire un événement. Par exemple, choisir un alphabet qui ne tient compte que de la machine émettrice afin de faire ressortir des chaînes d'événements à l'échelle des machines plutôt qu'à l'échelle des alarmes. Le DIG-DAG sera alors moins complet, mais peut donner des indices pour choisir les machines qu'il est important de

considérer. Ainsi, on pourra construire dans un second temps un DIG-DAG avec l'alphabet complet en se limitant à un jeu de machines plus restreints.

5.2 Évaluations expérimentales

5.2.1 Passage à l'échelle

Afin d'évaluer la capacité de passage à l'échelle de notre solution, nous construisons le DIG-DAG du log #0 issu d'une plateforme expérimentale, dont les erreurs sont déclenchées manuellement. Ce log contient 1409 événements (c'est-à-dire 2818 symboles) impliquant 27 machines (physiques comme virtuelles) et 29 types d'alarmes. La taille totale de l'alphabet est de 73. La topologie sous-jacente du système nous est inconnue.

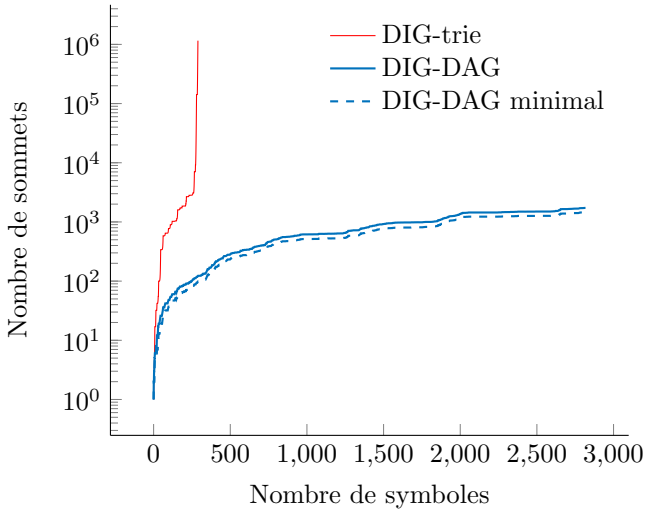


FIGURE 5.2 – Positionnement du DIG-DAG entre DIG-DAG minimal et DIG-trie.

La figure 5.2 compare la taille du DIG-DAG construit à partir du log #0, du DIG-trie correspondant et du DIG-DAG après minimisation *a posteriori* [Revuz, 1992]. La mémoire utilisée peut être estimée par le nombre de sommets de la structure. Nous pouvons observer que la taille du DIG-trie explose après 300 événements. La taille du DIG-DAG diffère quant à elle peu de celle du DIG-DAG minimal. Le rapport de taille entre les deux structures est en effet de 1.18.

La figure 5.3 montre le temps cumulé passé pour la construction du DIG-DAG du log #0

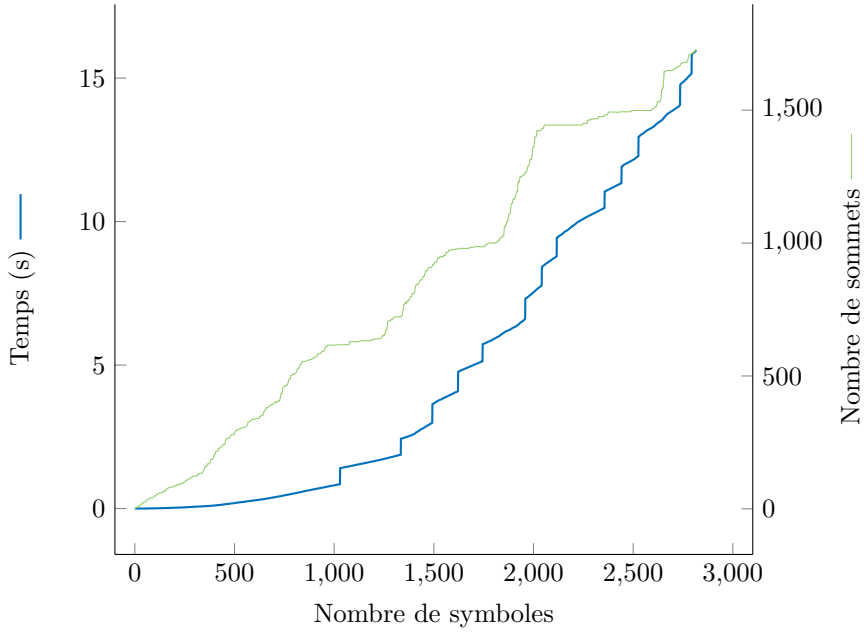


FIGURE 5.3 – Évolutions du temps de construction et du nombre de sommets du DIG-DAG en fonction de la taille du mot reçu.

ainsi que la croissance de la structure en fonction du nombre d'événements traités. Comme le DIG-DAG contient tous les motifs observés, il contient aussi les plus longs. De fait, la taille du DIG-DAG est croissante avec la taille du mot d'entrée. Borner le DIG-DAG en taille est envisageable mais a mené à des résultats insatisfaisants : par exemple, limiter la taille de la structure en contraignant la longueur du plus court chemin de la racine vers chaque sommet est insuffisant puisque le DIG-DAG contient un grand nombre de raccourcis. Finalement, le système de requête actuel permet d'extraire des motifs simples et pertinents du DIG-DAG non tronqué.

Influence des connaissances *a priori* sur la taille du DIG-DAG

Pour mieux décrire la pertinence de l'intégration de connaissances *a priori*, nous avons implémenté un simulateur à événements discrets afin de produire des logs d'alarmes artificiels. En effet, cela nous permet d'avoir une connaissance parfaite de la topologie du réseau à l'origine des logs. Informellement, notre générateur de logs est markovien et repose sur le graphe causal illustré en figure 5.4.

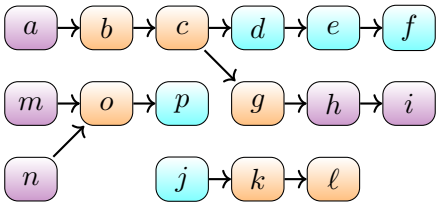


FIGURE 5.4 – Modèle du générateur de logs artificiels utilisés en section 5.2.2.

Chaque symbole se voit assigner un sommet. Les symboles a, h, i, m, n sont émis par la machine m_1, les symboles b, c, g, k, l, o par m_2 et d, e, f, j, p par m_3.

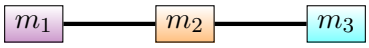


FIGURE 5.5 – Exemple jouet de réseau, qui peut être utilisé pour définir une relation spatiale de causalité.

Les liens de causalité respectent la topologie illustrée par la figure 5.5, similaire à celle introduite dans le chapitre 3 (les alarmes émises par les machines sont toutefois différentes). En particulier, la relation $\mathcal{C}$ est définie à partir de cette topologie : une alarme émise par m_1 ne peut directement causer une alarme sur m_3 et vice versa. Cela écarte 21% des causalités potentielles à considérer. Chaque symbole σ de $\Sigma \cup \overline{\Sigma}$ est doté d'une horloge exponentielle de paramètre λ_σ générant des débuts et des fins d'événements lorsque c'est autorisé. Une alarme source (a, m, n ou encore j) peut être déclenchée lorsqu'elle n'est pas déjà active. Par exemple, b peut être déclenchée seulement lorsque a est active. De même, o peut être déclenchée lorsque m ou n est active. Le motif $i \to k \to l$ est arbitrairement deux fois plus fréquent que les autres.

À partir d'un tel log, la figure 5.6 représente ainsi l'évolution de la taille du DIG-DAG construit avec et sans connaissance *a priori*. Autrement dit, nous comparons les situations où les causalités valides sont soit $\mathcal{C}$ (voir figure 5.4), soit Σ^2 tout entier. Le DIG-DAG intégrant *a priori* la topologie du réseau est 40% plus petit que celui pour lequel aucune hypothèse n'est faite *a priori*. Ce phénomène s'explique par le fait que réduire le nombre de causalités potentielles valides limite nécessairement la croissance du DIG-DAG. Dans le cas du DIG-DAG minimal, les gains obtenus grâce aux connaissances *a priori* sont similaires. Par ailleurs, ces courbes montrent également que le DIG-DAG passe bien à l'échelle sur ce log jouet, en termes de mémoire comme de temps de calcul.

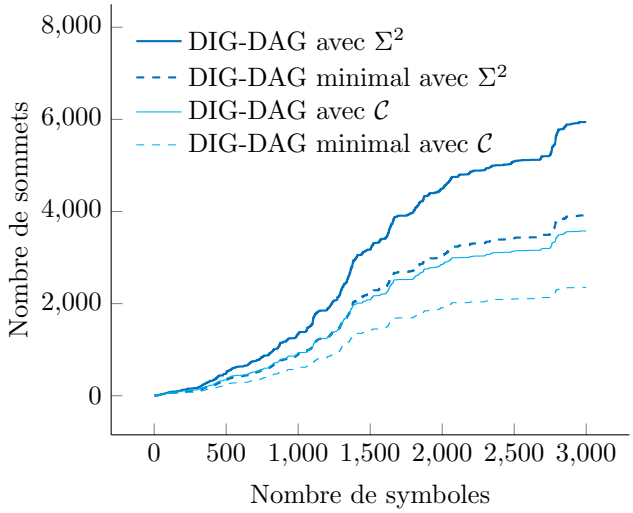


FIGURE 5.6 – Importance des connaissances *a priori* sur la taille du DIG-DAG.

5.2.2 Précision du diagnostic de fautes

Notre approche repose sur le stockage des chaînes de causalité. Dans cette section, nous illustrons la capacité du DIG-DAG à inférer le modèle causal sous-jacent de la figure 5.4. Nous utilisons l'approche présentée en section 5.1.2 et extrayons le sous-DIG-DAG tel que tous les arcs aient un ratio supérieur à ρ. Nous notons $E'(\rho)$ l'ensemble des arcs du sous-DIG-DAG résultant et E^m l'ensemble des arcs du modèle causal.

Nous comparons alors les deux ensembles. Pour cela, nous utilisons les métriques de *précision* $P(\rho)$ et de *rappel* $R(\rho)$:

$$P(\rho) = \frac{|E^m \cap E'(\rho)|}{|E'(\rho)|} \text{ et } R(\rho) = \frac{|E^m \cap E'(\rho)|}{|E^m|}.$$

Informellement, le rappel indique la proportion d'arcs du modèle absents dans le DIG-DAG, tandis que la précision indique la proportion d'arcs présents à tort dans le DIG-DAG.

Nous construisons un DIG-DAG à partir d'un log de longueur 1000 généré selon le modèle illustré en figure 5.4. Nous mesurons alors la précision et le rappel comme des fonctions de ρ.

La précision augmente avec ρ, tandis que le rappel est une fonction décroissante. En

effet, plus la seuil ρ augmente et plus le filtrage est sélectif : en augmentant celui-ci, il est plus facile d'oublier des arcs pertinents comme il est également plus difficile d'introduire des arcs à tort.

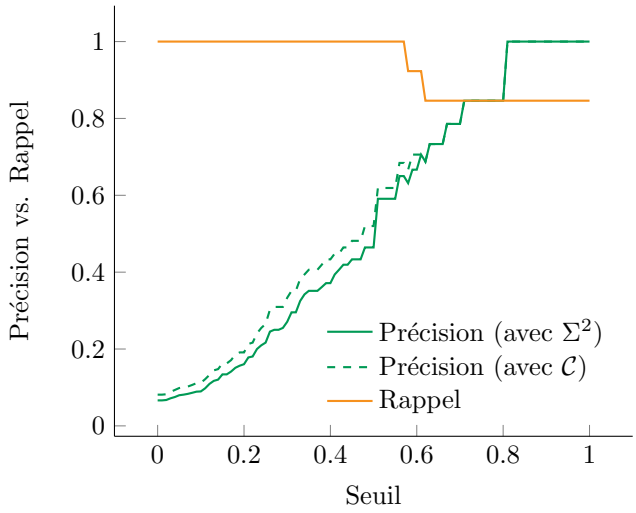


FIGURE 5.7 – Évolutions du temps de construction et du nombre de sommets du DIG-DAG en fonction de la taille du mot reçu.

Sur la figure 5.7, nous observons que la précision peut atteindre 0.65 et un rappel égal à 1 avec un seuil valant $\rho = 0.55$. Ainsi, le ratio $r_{u,\sigma}$ est un indicateur de la pertinence des causalités potentielles observées.

Pour de petits seuils, le rappel est égal à 1 : le log est suffisamment long pour que toutes les causalités aient pu être observées et stockées dans le DIG-DAG. Le rappel décroit pour les plus grandes valeurs de ρ, ce qui veut dire que des dépendances pertinentes peuvent tout de même être écartées. Dans notre exemple, comme o peut être une conséquence de m comme de n, l'alarme o peut apparaître sous des contextes différents. Par conséquent, les ratios $r_{g(u_0,m),o}$ et $r_{g(u_0,n),o}$ sont strictement inférieurs à 1 (légèrement supérieurs à 0.5 en l'occurrence). Dans tous les autres cas, les ratios obtenus dans le cadre de notre exemple jouet valent 1. Finalement, on peut remarquer qu'un rappel inférieur 1 n'est pas problématique pour l'analyse de fautes : il suffit de regarder les prédécesseurs actifs des sommets actifs du DIG-DAG étiquetés par o. Cela permet de différencier m de n dans la plupart des cas (soit m actif, soit n actif).

Ajouter de l'information *a priori* à travers $\mathcal{C}$ augmente la précision uniquement lorsque $\rho \leq 0.6$. En fait, la plupart des faux positifs pour des seuils plus grands sont des raccourcis

de chemins de causalité introduits par le modèle. La redéfinition de la relation spatiale n'a donc que peu d'effet sur la précision dans ce cas. Les alarmes distantes (du point de vue du modèle) sont peu co-occurrentes dans l'ensemble. Ainsi, si l'introduction de connaissances *a priori* améliore faiblement la précision, son intérêt principal réside en l'amélioration du passage à l'échelle du DIG-DAG en écartant d'office des causalités non pertinentes.

5.3 Exemples d'analyses de logs

Dans cette section, nous illustrons sur un exemple réel comment appliquer la méthodologie de la section 5.1.

Pour cela, nous disposons de trois logs, #1, #2 et #3, issus de réseaux GSM. Ces logs étant privés, ils ont été anonymisés. Chaque ligne décrit un événement spécifique : il contient l'identifiant de l'alarme, le nom de la machine émettrice, les dates de début et de fin ainsi que la sévérité de l'événement. La sévérité indique l'importance de l'alarme, valant 0 lorsqu'il s'agit d'un message informatif, 1 pour un avertissement, 2 pour les alarmes sévères et 3 pour les pannes majeures. Chaque triplet `(machine, alarm-id,severity)` correspond à un symbole comme expliqué en section 5.1.1.

Le tableau 5.1 donne pour chaque log sa taille et celle de l'alphabet correspondant. Les topologies du réseau sont inconnues.

	Log #1	Log #2	Log #3
Durée (h)	1042	48	330
Nb. d'entrées (total)	35,905	6,873	5,591
Nb. d'éléments réseaux	115	142	41
Nb. d'alarmes	43	66	39
Taille de l'alphabet	242	429	113

TABLE 5.1 – Dimensions des jeux de données bruts.

5.3.1 Simplification de l'alphabet et construction du DIG-DAG

Tout d'abord, une relation de causalité potentielle est définie par rapport à la nature ponctuelle des événements. Comme les logs peuvent durer plusieurs jours, nous considérons qu'un événement peut être la cause d'un autre s'il est apparu au plus une heure avant. Nous estimons ce critère suffisant pour capturer les causalités pertinentes.

Afin de limiter la taille du DIG-DAG, les alarmes co-occurrentes sont regroupées comme expliqué en section 5.1.4 avec paramètre $\alpha = 0.3$. De plus, nous écartons les clusters non pertinents, c'est-à-dire ceux dont la période d'activité totale est supérieure à 24 heures.

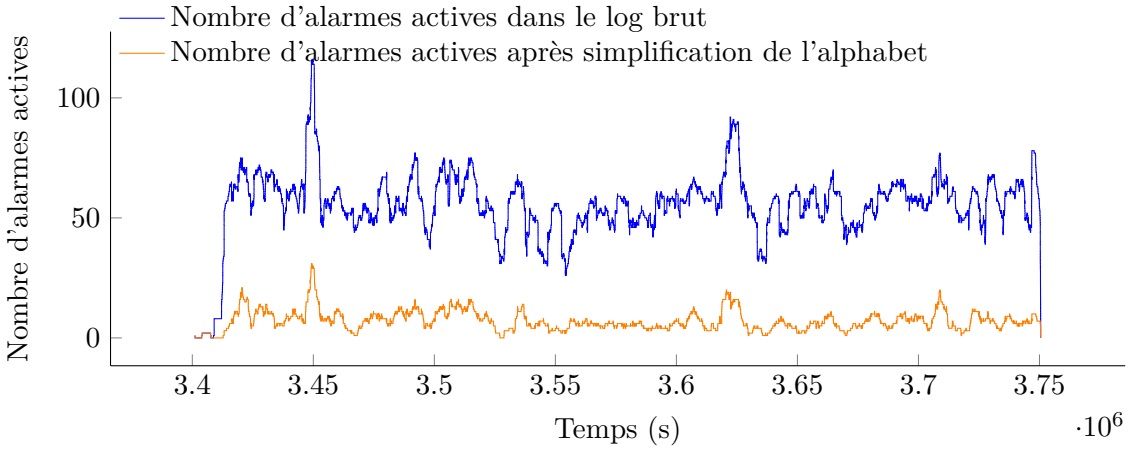


FIGURE 5.8 – Nombre d'alarmes actives en fonction du temps (logs bruts et simplifiés) au sein du log 1.

L'activité du log #1, avant et après simplification de l'alphabet, est représentée en figure 5.8. Il est possible de vérifier que de telles simplifications n'ont pas altéré le comportement observé du log original.

La taille des logs après simplification est montrée dans le tableau 5.2. Notons que passer l'étape de simplification de l'alphabet pendant l'analyse de cause racine mène à la construction de structures plus grandes. Par exemple, le log #1 est trop grand pour être ingéré directement et rapidement.

	Log #1	Log #2	Log #3
Nb. d'entrées après clustering	1,095	548	480
Nb. d'entrées après filtrage du spam	635	537	226
Nb. de clusters	103	167	69
Temps d'exécution du clustering	2.5s	3.7s	0.4s

TABLE 5.2 – Dimensions des logs après simplification de l'alphabet et temps d'exécution du clustering.s.

5.3.2 Utilisation de la topologie

Nous n'avons pas d'information sur la topologie du réseau qui a mené aux logs #1, #2 et #3, nous possédons toutefois un quatrième log réel (#4) dont la topologie est partiellement connue. En effet, les positions géographiques des machines sont données. Malheureusement, aucune vérité terrain n'est accessible pour ce log.

Afin de limiter la construction de causalités potentielles dans le DIG-DAG, nous pouvons ainsi écarter celles qui impliqueraient des machines trop éloignées les unes des autres (voir figure 5.9) lors de l'analyse du log #4. Il s'agit d'une vision grossière de la topologie du réseau, mais cela permet d'alléger considérablement la structure.

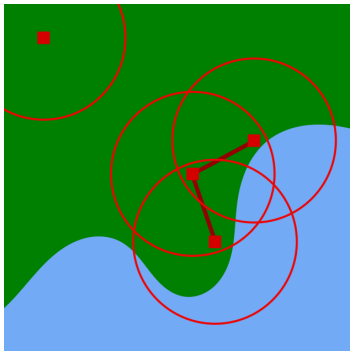


FIGURE 5.9 – Schéma : une topologie grossière peut être définie à partir des positions des machines (carrés rouges) en définissant arbitrairement un rayon d'interaction (cercles rouges). Les liens rouges entre les machines montrent les connexions possibles.

	Log #1	Log #2	Log #3
Nb. de sommets/arcs du DIG-DAG	20,540 / 443,496	4,364 /177,973	190 / 643
Nb. de sommets/arcs après requête ($\rho = 0.7$)	19 / 35	142 / 996	34 / 54
Temps de construction du DIG-DAG	25.1s	8.9s	38ms
Temps d'exécution de la requête	0.7s	1.0s	42ms

TABLE 5.3 – Performances de notre système de requête.

5.3.3 Requêtes sur le DIG-DAG

Malgré leurs constructions rapides, les DIG-DAG sont difficilement lisibles par un être humain (voir tableau 5.3). Nous extrayons à présent les motifs menant à des erreurs critiques. Pour chaque jeu de données, nous employons les requêtes avec les paramètres suivant :

- S est restreint aux *sommets pertinents* qui correspondent aux clusters contenant au moins un événement dont la sévérité est supérieure ou égale à 1 ;
- T est restreint aux *sommets critiques* qui correspondent aux clusters contenant au moins un événement de sévérité 2 ou 3 ;
- P_v est définie pour ne considérer que des sommets pertinents.
- P_e est définie pour ne considérer que des arcs dont le ratio est supérieur à $\rho \in [0, 1]$;
- $\mathcal{M}$ accepte n'importe quel motif.

5.3.4 Pertinence des requêtes

Comme nous n'avons ni connaissance *a priori* ni expertise à propos des logs étudiés, les motifs décrits sont particulièrement génériques. Les deux paragraphes suivants montrent cependant que ces requêtes naïves permettent quand même d'améliorer la lisibilité des DIG-DAG et exhibent des informations pertinentes au sujet des pannes.

Capacité de filtrage des requêtes

Ce paragraphe présente des résultats pour différentes valeurs de ρ.

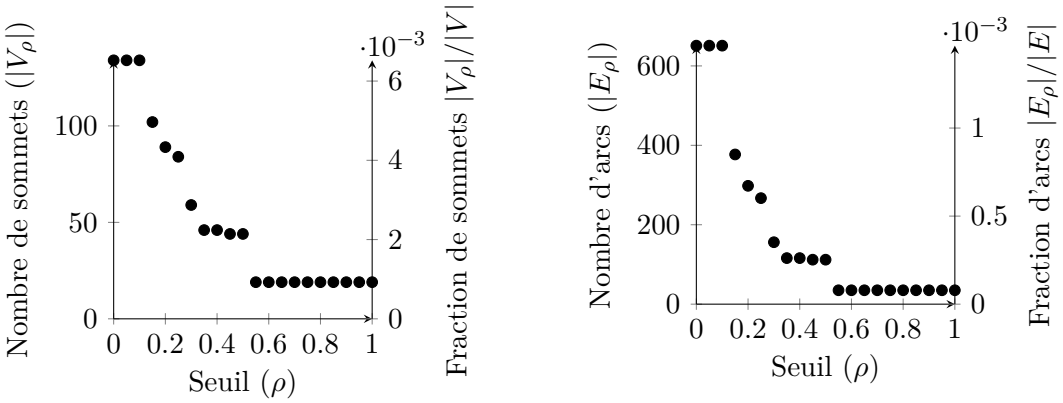


FIGURE 5.10 – Nombre de sommets$|V_\rho|$ (resp. d'arcs $|E_\rho|$) des sous-DIG-DAG (d'après le log 1) filtrants par rapport au seuil ρ (axe de gauche). Ce nombre est divisé par le nombre de sommets $|V|$ (resp. d'arcs $|E|$) du DIG-DAG tout entier (axe de droite).

Dans la figure 5.10, pour tout ρ, les requêtes gardent une faible fraction du DIG-DAG construit à partir du log #1. Cela souligne l'efficacité du système de requête pour extraire des motifs depuis le DIG-DAG. Pour $\rho \geq 0.7$, les sous-DIG-DAG correspondants ont moins de 19 sommets et 35 arcs. Ils sont ainsi suffisamment petits pour être lisible par un être humain. Le choix du seuil ρ est un compromis entre lisibilité et quantité d'information.

Précision des requêtes

Pour chaque jeu de données, les experts ayant fourni les logs nous ont également donné les causes racines qu'ils avaient identifiées (indépendamment de nos travaux). Lorsque $\rho = 0.7$, nous avons pu observer que les sous-DIG-DAG obtenus contiennent à chaque fois les causes racines identifiées. De plus, un sous-DIG-DAG fournit une structure plus riche qu'une simple cause racine. Par exemple, la figure 5.11 montre le sous-DIG-DAG obtenu à partir du log #1. La cause racine identifiée de ce log est l'événement `(77397912,4004,2)`, bel et bien apparente dans le sous-DIG-DAG.

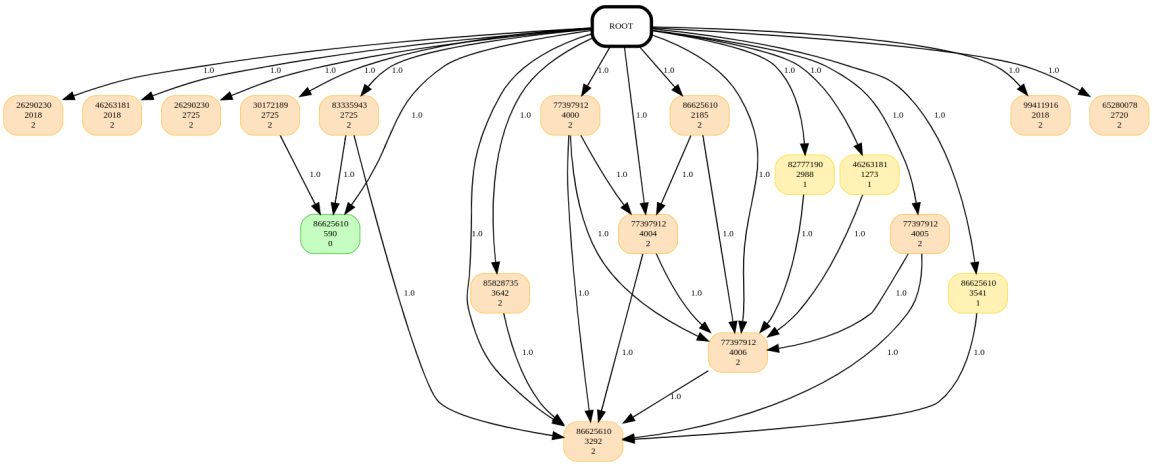


FIGURE 5.11 – Sous-DIG-DAG issu de la requête du log #1 ($\rho = 0.7$). Les étiquettes des sommets sont mentionnées sous la forme `(machine, id-alarme, sévérité)`. Les couleurs dépendent de la sévérité du représentant du cluster. Les ratios sont indiqués pour chaque arcs.

5.4 Conclusion

Premièrement, l'ensemble des outils présentés dans les chapitres 3 et 4 ont été implémentés à travers une librairie Python 3. Lors du développement, une véritable attention a été portée sur le *design* du code afin de rendre ce dernier le plus flexible possible. En effet, étant un travail de recherche, il est important d'avoir toute latitude pour y insérer des modifications ou des ajouts. Le travail d'implémentation a notamment mené à de riches échanges avec plusieurs *business units* de Nokia. Nous leur avons fourni un prototype opérationnel et avons pu profiter en contre-partie de logs réels ainsi que de leurs retours.

Par ailleurs, nous avons pu illustrer à travers ce chapitre l'intérêt de notre solution pour un problème d'analyse de cause racine. Tout d'abord car notre solution est interprétable : l'administrateur du réseau a le contrôle aussi bien sur la définition de la causalité que des motifs qu'il souhaite inspecter. De plus, il est possible d'utiliser le ratio comme la probabilité empirique d'observer une alarme dans un certain contexte, conditionnellement à l'observation de cette alarme sans contexte particulier. Il est ainsi possible de filtrer les motifs peu probables ou peu pertinents.

Désormais, étant donné un DIG-DAG et ses états actifs, nous souhaiterions à terme

prédire l'état du réseau. Si la construction en ligne du DIG-DAG est une condition nécessaire remplie pour la prédiction d'alarmes, la notion de contexte, sur laquelle est basé le ratio, demeure assez floue et gagnerait à être définie plus rigoureusement. Or, nous souhaitons faire de la prédiction d'alarmes conditionnellement au « contexte » courant. Il serait donc intéressant d'adapter la notion de ratio à la prédiction d'alarmes, ce qui est difficile sans un cadre théorique plus solide.

Finalement, une expertise humaine reste encore nécessaire dans le processus d'analyse de causes racines. Nous pensons toutefois que ce travail est un premier pas vers un diagnostic automatique et à terme vers des réseaux auto réparants.

Chapitre 6

Expressivité comparée des modèles HMM et RNN

Nous avons vu dans le chapitre 2 que deux types de modèles génératifs se distinguaient pour la prédiction de séries temporelles : HMM à états continus et RNN. Bien que ces deux modèles soient issus de communautés différentes, ils possèdent des similarités dans leur fonctionnement. Ainsi, ils modélisent tous deux une loi de probabilité sur des observations $x_{0:t}$ qui dépendent d'une suite d'états latents $h_{0:t}$. De plus, ces deux modèles font tous les deux apparaître une structure markovienne. Cela soulève une question naturelle : quel modèle choisir dans une application concrète ?

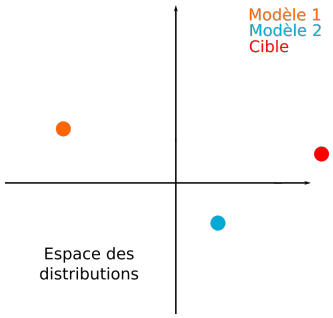
Un point important de comparaison est celui de l'*expressivité*. Cette notion décrit la capacité d'un modèle à approcher une grande variété de distributions. Du point de vue d'un utilisateur, ce critère est important dans le choix d'un modèle. En effet, il permet de déterminer *a priori* le modèle susceptible d'approcher au plus près la distribution sous-jacente aux données étudiées. Pour répondre à cette question, il existe deux angles d'attaque : empiriquement d'une part, et analytiquement d'autre part.

La méthode la plus courante pour comparer deux modèles est de les confronter à l'*expérience*. À partir d'un même jeu de données, chacun des modèles est entraîné puis testé sur ce jeu. Il est alors possible d'analyser leurs performances. Pour comparer leur expressivité, il suffit de déterminer le modèle qui approche le mieux les données observées. En effet, plus un modèle est expressif, plus il est susceptible de modéliser des distributions variées et donc

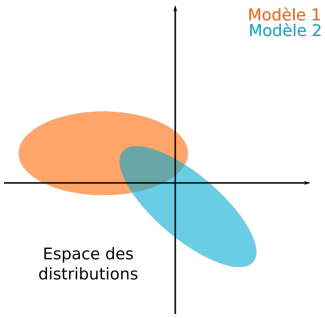
a priori proches de la distribution du jeu de données testé. Le choix du jeu de données permet d'explorer une grande diversité de cas d'usage. Le test peut porter sur des données artificielles comme des données réelles. Les jeux artificiels permettent de comparer les modèles sur des caractéristiques précises et choisies, tandis que les jeux de données réelles permettent de réaliser le test dans un contexte d'utilisation pratique. Toutefois, il est difficile de généraliser le résultat d'un test. Le fait qu'un modèle soit plus performant qu'un autre sur un certain type de données ne signifie pas qu'il est systématiquement plus performant. Par ailleurs, l'utilisation d'un modèle requiert une diversité de choix en termes de modélisation comme de paramétrage. En effet, il faut encore déterminer quels algorithmes d'entraînement, paramètres et hyperparamètres utiliser. Dans le cadre d'un test, ce sont autant de choix supplémentaires qui complexifient l'analyse. Par exemple, [Chung et al., 2014] réalise la comparaison empirique entre plusieurs types de réseaux récurrents : les RNN classiques, les GRU et les LSTM. Pour cela, ils confrontent ces trois modèles au problème de prédiction de parole et de musique à partir de jeux de données réels. Bien que les GRU et LSTM montrent plus de facilité à modéliser des dépendances à long terme qu'un RNN classique lors de ces expérimentations, ces travaux ne permettent pas de départager clairement GRU et LSTM. Dans le scénario où les performances des deux modèles comparés ne se distinguent pas assez, une approche expérimentale ne fournit pas assez d'éléments permettant de comprendre les différences entre les deux modèles.

Au contraire, une comparaison *analytique* permet de déterminer exactement l'ensemble des distributions qui peuvent être atteintes par chaque modèle. Une analogie pour comprendre la différence entre ces deux approches est la différence entre tests et méthodes formelles en vérification de systèmes. Un test permet de vérifier le bon fonctionnement d'un système sur un cas ponctuel, tandis que les méthodes formelles permettent de vérifier le bon fonctionnement du système à travers un système de règles [Monin, 2012]. À la différence d'un test, cette méthode de vérification n'est pas ponctuelle, elle est valide pour l'ensemble des cas définis par les règles. Toutefois, les contraintes induites par ces règles ne sauraient couvrir tous les cas envisageables lors de l'utilisation. Elles offrent ainsi une représentation limitée du système. Une comparaison analytique donne un point de vue similaire : dans la limite d'un cadre théorique défini, nous déterminons toutes les distributions modélisables par chaque modèle. La figure 6.1 illustre les deux approches décrites plus haut.

Dans ce chapitre, nous comparons analytiquement l'expressivité des HMM à états continus et des RNN. Notamment, nous définissons l'expressivité d'un modèle comme l'ensemble de toutes les lois $p_\theta(x_{0:t})$ offertes par le modèle quel que soit $\theta \in \Theta$ (l'ensemble des para-



(a) Approche empirique.



(b) Approche analytique.

FIGURE 6.1 – Illustration de la comparaison de l'expressivité de deux modèles (orange et bleu). Le plan représente l'espace des distributions dans lequel s'inscrit l'étude. Dans la figure 6.1a, le point rouge représente la distribution qui a servi à générer le jeu de données étudié, les points orange et bleu sont les distributions obtenues après avoir entraîné chaque modèle. Les modèles sont comparés à travers leur capacité à approcher la distribution cible. En pratique, cette distribution cible n'est pas connue de manière explicite. Dans la figure 6.1b, chaque surface (orange ou bleue) correspond à l'ensemble des distributions atteignables par le modèle correspondant. Les distributions considérées sont généralement définies au sein d'un cadre d'étude simplificateur. Ainsi, nous déterminons pour chaque distribution si elle peut être modélisée par un modèle, l'autre, les deux ou aucun.

mètres du modèle : par exemple, l'ensemble des poids et biais d'un réseau de neurones). Parmi les questions qu'une telle comparaison peut soulever, nous cherchons à déterminer s'il existe une inclusion, une intersection, etc. des ensembles de lois caractérisés par chaque modèle.

Afin de réaliser cette comparaison, nous introduisons tout d'abord en section 6.1 un modèle appelé GUM (pour *Generative Unified Model*) qui permet d'englober les deux modèles à partir de leurs similarités structurelles. Idéalement, nous souhaiterions caractériser les lois de probabilité que l'on peut modéliser avec chaque modèle par l'intermédiaire du GUM. Toutefois, cette caractérisation théorique est compliquée à mettre en œuvre dans un cadre très général. Pour pallier cette difficulté, nous introduisons un certain nombre de simplifications, notamment, nous nous restreignons au cas des GUM linéaires et gaussiens. Outre ces hypothèses simplificatrices, nous supposons dans un premier temps que les états cachés et les observations sont scalaires. Nous bénéficions alors d'un cadre d'étude favorable pour analyser rigoureusement en section 6.2 l'expressivité de ce modèle unificateur. Enfin, en section 6.3, nous cherchons à étendre cette étude en relâchant l'hypothèse d'unidimensionnalité des états cachés. Les travaux présentés dans la section 6.2 ont été publiés à ICMLA 2019

[Salaün et al., 2019b] ; les résultats présentés dans la section 6.3 font l'objet d'un article qui sera soumis prochainement.

6.1 HMM, RNN et modèles génératifs unifiés (GUM)

Comme expliqué dans le chapitre 2, nous rappelons que la loi jointe des observations modélisée par un HMM est la marginale d'une loi jointe $p(h_{0:t}, x_{0:t})$ qui s'écrit à partir de la donnée des lois $p(h_0)$, $p(h_{s+1}|h_s)$ et $p(x_s|h_s)$ pour tout s,

$$p(h_{0:t}, x_{0:t}) = p(h_0) \prod_{s=0}^{t-1} p(h_{s+1}|h_s) \prod_{s=0}^{t} p(x_s|h_s). \tag{6.1}$$

Par ailleurs, dans un RNN génératif, on construit directement la loi jointe des observations

$$\begin{aligned} p(x_{0:t}) &= p(x_0) \prod_{s=0}^{t-1} p(x_{s+1}|x_{0:s-1}) \\ &= p(x_0) \prod_{s=0}^{t-1} p(x_{s+1}|h_s), \end{aligned} \tag{6.2}$$

où h_s est fonction de $x_{0:s}$: $h_s = f(h_{s-1}, x_s)$ et $p(x_s|h_s)$ une loi donnée.

À une translation près des indices temporels utilisés pour les RNN (on considérera à présent $h_t \leftarrow h_{t-1}$), on remarque que les deux modèles sont caractérisés par la loi conditionnelle de x_t sachant h_t, mais que la construction des variables latentes h_t diffère : dans le premier cas, h_t est aléatoire et est caractérisé par la loi $p(h_t|h_{t-1})$; dans le second cas, h_t est une fonction déterministe de h_{t-1} et x_{t-1}. Dans les deux cas, la paire (h_t, x_t) est markovienne, c'est-à-dire

$$\begin{aligned} p(h_t, x_t|h_{0:t-1}, x_{0:t-1}) &= p(h_t, x_t|h_{t-1}, x_{t-1}) \\ &= p(h_t|h_{t-1}, x_{t-1}) p(x_t|h_t, h_{t-1}, x_{t-1}). \end{aligned} \tag{6.3}$$

Notamment, dans les deux modèles,

$$p(x_t|h_t, h_{t-1}, x_{t-1}) = p(x_t|h_t), \tag{6.4}$$

mais,

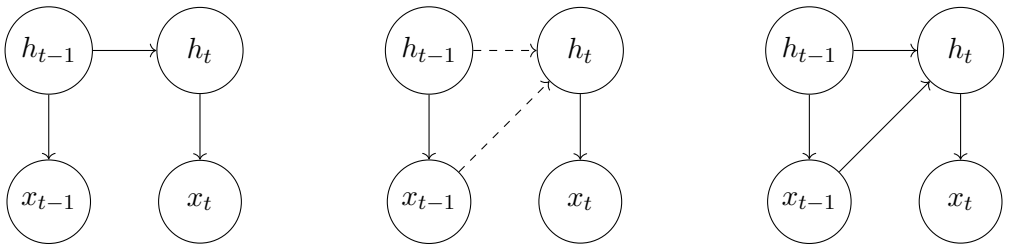
$$\text{pour un HMM : } p(h_t|h_{t-1}, x_{t-1}) = p(h_t|h_{t-1}); \tag{6.5}$$

$$\text{pour un RNN : } \; p(h_t|h_{t-1}, x_{t-1}) = \delta_{f(h_{t-1}, x_{t-1})}(h_t) \times p(x_t|h_t), \tag{6.6}$$

(où δ est une fonction de Dirac). Finalement, HMM et RNN sont un cas particulier d'un même modèle qu'on appellera modèle génératif unifié (*Generative Unified Model* ou GUM) dans lequel

$$p(h_t, x_t|h_{t-1}, x_{t-1}) = p(h_t|h_{t-1}, x_{t-1})p(x_t|h_t), \tag{6.7}$$

et dont les représentations graphiques sont données dans la figure 6.2.



(a) HMM (b) RNN (c) GUM

FIGURE 6.2 – Dépendances conditionnelles dans les RNN, HMM, et GUM. Les arcs en pointillés représentent les dépendances déterministes. Les arcs pleins représentent les dépendances probabilistes.

Afin de comparer la loi des observations $p(x_{0:t})$ induite par chacun des modèles HMM et RNN, nous nous plaçons dans le cadre général du GUM, dans lequel la loi jointe des observations est une marginale de $p(h_{0:t}, x_{0:t}) = p(h_0) \prod_{s=1}^{t} p(h_s|h_{s-1}, x_{s-1})p(x_s|h_s)$. Bien évidemment, $p(x_{0:t}) = \int_{h_{0:t}} p(h_{0:t}, x_{0:t}) \mathrm{d}h_{0:t}$ n'est pas, dans la plupart des cas, calculable analytiquement. Pour pouvoir analyser $p(x_{0:t})$, (et donc considérer les deux cas particuliers HMM et RNN), nous nous plaçons dans un cadre simplificateur :

- dans ce chapitre, nous nous plaçons dans le cadre linéaire et gaussien afin de comprendre plus facilement l'impact sur la loi des observations de transitions déterministes par rapport à des transitions stochastiques ;
- dans la section 6.2, nous nous plaçons dans le cas où les variables latentes sont scalaires ;

- plus tard, dans la section 6.3, nous relâchons cette hypothèse (cas multidimensionnel).

Par ailleurs, il est à noter que nous ne nous intéresserons pas aux problèmes d'apprentissage correspondants aux modèles étudiés (nous avons néanmoins vu dans le chapitre 2 comment des modèles similaires étaient entraînés).

6.2 GUM linéaires et gaussiens : le cas unidimensionnel

À partir de maintenant, nous considérons un GUM linéaire et gaussien, c'est-à-dire un modèle dans lequel la loi initiale, les lois de transition et celles d'émission s'écrivent pour tout t, [1]

$$p(h_0) = \mathcal{N}(h_0; 0; \eta), \tag{6.8}$$

$$p(h_t|h_{t-1}, x_{t-1}) = \mathcal{N}(h_t; ah_{t-1} + cx_{t-1}; \alpha), \tag{6.9}$$

$$p(x_t|h_t) = \mathcal{N}(x_t; bh_t; \beta). \tag{6.10}$$

Dans le modèle GUM décrit ci-dessus, la loi jointe des observations est alors une gaussienne multivariée par extension du cas linéaire et gaussien (voir section 2.2.2) que nous nous proposons d'étudier à présent. Sans perte de généralité, nous imposons que

$$\forall t \in \mathbb{N}, p(x_t) = \mathcal{N}(x_t; 0; 1), \tag{6.11}$$

de manière à analyser $p(x_{0:t})$ sur la base de sa matrice de covariance. Dans la suite, nous commençons par identifier tous les GUM (équations (6.8) à (6.10)) qui satisfont la contrainte (6.11). De cette identification est déduite la matrice de covariance Σ_t de la loi jointe $p(x_{0:t})$, que nous exploitons. Plus précisément, nous allons montrer qu'il est possible de modéliser par un GUM n'importe quelle distribution normale centrée réduite dont la matrice de covariance est Toeplitz géométrique (voir section 6.2.2).

Ces hypothèses ne sont pas nécessairement réalistes d'un point de vue pratique. Par exemple, un RNN fait toujours intervenir une fonction d'activation non linéaire en pratique. Toutefois, il est toujours possible de considérer des GUM intégrant des lois de probabilités plus complexes pour $p(h_t|h_{t-1}, x_{t-1})$ que celles étudiées dans ce chapitre. Par exemple,

1. Pour rappel, la notation $\mathcal{N}(x; \mu; \sigma^2)$ représente la densité de probabilité de la loi normale de moyenne μ et de variance σ^2 considérée comme fonction de la variable x.

cela pourrait être une transition de la forme $p(h_t|h_{t-1}, x_{t-1}) \overset{\text{ex.}}{=} \mathcal{N}(h_t; f(h_{t-1}, x_{t-1}); \sigma^2)$, avec $f : (h, x) \mapsto \max(ah + cx, 0)$ (un *ReLu*) et a et c deux poids. En ce sens, nous comparons, à hypothèses égales, nos différents modèles. Alors, l'analyse des sous-cas ($c = 0$) et ($\alpha = 0 \wedge c^2 = \eta$) nous permet de dresser une comparaison des HMM et RNN, pour la modélisation d'observation dans le cas linéaire et gaussien.

6.2.1 Calculs préliminaires

Avant toute chose, rappelons une relation classique :

Lemme 5 *Pour tout $m, n \in \mathbb{N}$, soient $a \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^n, c \in \mathbb{R}^m, \alpha \in \mathbb{R}^{m\times m}, \beta \in \mathbb{R}^{n\times n}$. Alors, pour tout $y \in \mathbb{R}^m$,*

$$\int_{x\in\mathbb{R}^n} \mathcal{N}(y; ax + c; \alpha)\mathcal{N}(x; b; \beta)dx = \mathcal{N}(y; ab + c; \alpha + a\beta a^T)$$

où a^T est la matrice transposée de a. De plus, ce résultat s'étend aisément au cas déterministe (α et/ou β nuls).

Modèle GUM contraint

Nous cherchons un GUM vérifiant la contrainte (6.11), c'est-à-dire un jeu de paramètres $\theta = \{a, b, c, \eta, \alpha, \beta\} \in \mathbb{R}^3 \times \mathbb{R}^{+3}$ tel que la loi jointe $p_\theta(x_{0:t})$ satisfasse (6.11). Partant de $p(h_0) = \mathcal{N}(h_0; 0; \eta)$, nous avons

$$\begin{aligned} p(x_0) &= \int_{h_0\in\mathbb{R}} p(x_0|h_0)p(h_0)\mathrm{d}h_0 \\ &= \mathcal{N}(x_0; 0; \beta + b^2\eta) \text{ d'après le corollaire 5,} \end{aligned} \tag{6.12}$$

et donc, d'après la contrainte (6.11), $\beta = 1 - b^2\eta$. De même,

$$\begin{aligned} p(x_1) &= \iiint p(x_1|h_1)p(h_1|h_0, x_0)p(x_0|h_0)p(h_0)\mathrm{d}h_1\mathrm{d}h_0\mathrm{d}x_0 \\ &= \int_{h_1\in\mathbb{R}} \mathcal{N}(x_1; bh_1; \beta) \int_{h_0\in\mathbb{R}} \mathcal{N}(h_0; 0; \eta) \int_{x_0\in\mathbb{R}} \mathcal{N}(x_0; bh_0; \beta)\mathcal{N}(h_1; ah_0 + cx_0; \alpha)\mathrm{d}x_0\mathrm{d}h_0\mathrm{d}h_1 \\ &= \mathcal{N}(x_1; 0; \beta + b^2(\alpha + \beta c^2 + (a + bc)^2\eta)) \text{ d'après le corollaire 5.} \end{aligned}$$

Finalement, grâce à la contrainte (6.11) et la relation $\beta = 1 - b^2\eta$, nous obtenons $\alpha = (1 - a^2 - 2abc)\,\eta - c^2$.

En résumé, l'ensemble des GUM vérifiant la contrainte (6.11) et paramétrés par $\theta = \{\eta, a, c, \alpha, b, \beta\}$ doivent vérifier

$$\beta = 1 - b^2\eta, \tag{6.13}$$

$$\alpha = \Big(1 - a^2 - 2abc\Big)\,\eta - c^2. \tag{6.14}$$

Par ailleurs, puisque ce sont des variances, ces deux quantités doivent être positives.

Notons que le modèle GUM englobe un HMM et un RNN particuliers :

- lorsque $c = 0$, nous obtenons un HMM linéaire et gaussien (voir section 2.2.2) : l'équation (6.14) devient alors $\alpha = (1 - a^2)\eta$;
- lorsque $\alpha = 0$, nous obtenons un RNN dont la fonction d'activation est linéaire. Par abus de notations, l'équation (6.9) devient en effet une transition déterministe. De plus, dans les conventions RNN, on se donne en pratique une première observation $p(x_{-1})$ (qui est supposée être une gaussienne centrée réduite dans le cadre de la contrainte (6.11)) et le vecteur latent h_{-1} est initialisé à zéro. Cette étape d'initialisation d'un RNN n'apparaît pas telle quelle dans un GUM mais est plutôt retranscrite à travers la contrainte $\eta = c^2$. Ainsi, l'équation (6.13) devient $\beta = 1 - (bc)^2$ et l'équation (6.14) devient $0 = c^2 a(a + 2bc)$.

Calcul de la matrice de covariance de la loi jointe $p(x_{0:t})$

D'après les hypothèses (6.8)-(6.10) et la contrainte (6.11), nous savons que la loi jointe $p(x_{0:t})$ est une loi normale multivariée de vecteur moyenne nul et de matrice de covariance Σ_t dont les termes diagonaux sont égaux à 1 :

$$\forall t \in \mathbb{N}, p(x_{0:t}) = \mathcal{N}(x_{0:t}; \mathbf{0}_t, \Sigma_t). \tag{6.15}$$

Par ailleurs, les termes non diagonaux de cette matrice de covariance sont définis par les covariances

$$\forall t \in \mathbb{N}, \forall k \in \mathbb{N}^*,\ \mathrm{Cov}(x_t, x_{t+k}) = (a+bc)^{k-1}(ab^2\eta + bc). \tag{6.16}$$

Preuve. Prouvons le résultat (6.16) ci-dessus.

$$\begin{aligned}
\forall t \in \mathbb{N}, k \in \mathbb{N}^*,\ \mathrm{Cov}(x_t, x_{t+k}) &= b\ \mathrm{Cov}(x_t, h_{t+k}) \\
&= a\,(b\ \mathrm{Cov}(x_t, h_{t+k-1})) + bc\ \mathrm{Cov}(X_t, x_{t+k-1}) \\
&= (a+bc)\ \mathrm{Cov}(x_t, x_{t+k-1}) \\
&= (a+bc)^{k-1}\ \mathrm{Cov}(x_t, x_{t+1}) \\
&= (a+bc)^{k-1}(ab^2\eta + bc).
\end{aligned}$$

□

Notons que l'expression de la covariance $\mathrm{Cov}(x_t, x_{t+k})$ explicitée dans (6.16) montre un comportement du GUM déjà bien observé dans les HMM et RNN ; à moins que $|a+bc| = 1$, les observations courantes sont géométriquement décorrélées des observations passées ; dans le cas où $|a+bc| = 1$, (6.14) et (6.13) donnent $\alpha = 0$ et $\beta c^2 = 0$. En d'autres termes, pour obtenir des dépendances au long terme, le déterminisme à travers le temps est requis. Ce phénomène a déjà été observé dans [Bengio and Frasconi, 1995].

6.2.2 Matrices de covariance Toeplitz géométriques

Comme nous l'avons vu à travers l'équation (6.16), pour tout GUM linéaire gaussien sous contrainte (6.11), la covariance $\mathrm{Cov}(x_t, x_{t+k})$ est géométrique, c'est-à-dire,

$$\exists (A, B) \in \mathbb{R}^2 | \forall t \in \mathbb{N}, \forall k \in \mathbb{N}^*,\ \mathrm{Cov}(x_t, x_{t+k}) = A^{k-1}B.$$

Réciproquement, pour tout $A, B \in \mathbb{R}$, la matrice de Toeplitz [2] symétrique $\boldsymbol{R}_t(A, B)$ dont la première ligne est définie par $[1, B, AB, A^2B, ..., A^{t-1}B]$ n'est pas nécessairement une matrice de covariance. En effet, nous ne savons pas si $\boldsymbol{R}_t(A, B)$ est semi-définie positive. Nous caractérisons donc le domaine (A, B) pour lequel $\boldsymbol{R}_t(A, B)$ est une matrice de covariance pour tout $t \in \mathbb{N}^*$. Nous avons le résultat suivant.

2. Une matrice de Toeplitz est une matrice dont les coefficients sont égaux sur une même diagonale.

Théorème 6 *Soit $\boldsymbol{R}_t(A,B)$ la matrice de Toeplitz symétrique dont la première ligne est*

$$[1, B, AB, ..., A^{t-1}B].$$

$\boldsymbol{R}_t(A,B)$ est une matrice de covariance pour tout $t \in \mathbb{N}^*$ si et seulement si (A,B) appartient au parallélogramme $\mathcal{P}$ défini par

$$A \in [-1,1] \text{ et } \frac{A-1}{2} \leq B \leq \frac{A+1}{2},$$

ou la droite $\mathcal{D}$ définie par

$$B = 0.$$

Nous notons $\mathcal{S} = \mathcal{P} \cup \mathcal{D}$.

Preuve. Commençons par restreindre le domaine (A,B) des points tels que $\boldsymbol{R}_t(A,B)$ est semi-définie positive pour tout t. Pour ce faire, remarquons tout d'abord que si M est une matrice semi-définie positive de dimension n, toute sous-matrice $M_{i_1,\cdots,i_k}$ de M obtenue en ne conservant que les lignes et colonnes d'indices $1 \leq i_1, \cdots, i_k \leq n$ est également semi-définie positive. Ainsi, les matrices de Toeplitz dont la première ligne est $[1,B]$ et respectivement $[1,A^tB]$ (quel que soit t) sont nécessairement semi-définies positives pour tout $t \in \mathbb{N}$, ce qui implique que $|B| \leq 1$ et respectivement $|A| \leq 1$ si $B \neq 0$. Nous cherchons donc pour quels $(A,B) \in [-1,1]^2 \cup \{(A,0); A \in \mathbb{R}\}$, la matrice $\boldsymbol{R}_t(A,B)$ est pour tout $t \in \mathbb{N}$ semi-définie positive.

Rappelons que dans le cas stationnaire, la fonction de $\mathbb{N}$ dans $\mathbb{R}$ qui à tout k associe c_k est une fonction de covariance (ou encore une suite de covariance) si et seulement si pour tout $n \geq 0$, la matrice de Toeplitz de première ligne $[c_0, ..., c_n]$ est une matrice de covariance, c'est-à-dire semi-définie positive. Cet ensemble de contraintes restreint donc l'ensemble des suites possibles : $\{c_0, c_1, c_2, ...\}$ est une fonction de covariance si et seulement si $c_0 \geq 0$, $\begin{bmatrix} c_0 & c_1 \\ c_1 & c_0 \end{bmatrix} \geq 0$ (et donc $|c_1| \leq c_0$), $\begin{bmatrix} c_0 & c_1 & c_2 \\ c_1 & c_0 & c_1 \\ c_2 & c_1 & c_0 \end{bmatrix} \geq 0$, etc. Il est possible d'exprimer cette contrainte en termes fonctionnels : le théorème de Carathéodory-Toeplitz [Akhiezer and

Kemmer, 1965] affirme que $\{c_0, c_1, c_2, ...\}$ est une fonction de covariance si et seulement si

$$C(z) = c_0 + 2\sum_{k=1}^{+\infty} c_k z^k,$$

est une fonction de la classe de Carathéodory, *ie.* $C(z)$ a une partie réelle positive pour z dans le disque unité ouvert. Ainsi, nous cherchons pour quelles valeurs de A et de B la fonction

$$1 + 2\sum_{k=1}^{+\infty}(A^{k-1}B)z^k$$

appartient à une telle classe. En écrivant $z = re^{i\theta}$, cette appartenance se traduit par la vérification de la contrainte

$$1 + 2(B - A)\cos(\theta)r - A(2B - A)r^2 \geq 0$$

pour tout $r \in [0, 1[$ et $\theta \in [-\pi, \pi[$. Puisque cette condition devrait être vérifiée pour tout θ (et donc pour les valeurs de θ qui minimisent la quantité $(B - A)\cos(\theta)r$), il est suffisant de chercher les A et B tels que

$$\Xi_{A,B}(r) = 1 - 2|B - A|r - A(2B - A)r^2 \geq 0,$$

pour tout $r \in [0, 1[$.

Supposons tout d'abord que $A(2B - A) = 0$, alors

- si $A = 0$,
$$\Xi_{A,B}(r) \geq 0 \text{ si et seulement si } |B| \leq \frac{1}{2};$$
- si $2B - A = 0$, nous avons également
$$\Xi_{A,B}(r) \geq 0 \text{ si et seulement si } |B| \leq \frac{1}{2}.$$

Supposons désormais $A(2B - A) > 0$. Alors, par concavité du polynôme $\Xi_{A,B}$:

$$\Xi_{A,B}(r) \geq \min(\Xi_{A,B}(0), \Xi_{A,B}(1)) = 1 - 2|B - A| - A(2B - A).$$

Ainsi, nous avons besoin de caractériser l'ensemble

$$\{(A, B); 1 - 2|B - A| - A(2B - A) \geq 0\}.$$

Si $B - A \geq 0$ (resp. $B - A < 0$), cette condition se réécrit $B \leq \frac{A+1}{2}$(resp. $B \geq \frac{A-1}{2}$).

Considérons finalement le cas $A(2B - A) < 0$. Dans ce cas, $\Xi_{A,B}$ est convexe et possède un discriminant positif (égal à $4B^2$). Il suffit donc d'étudier la position des racines de $\Xi_{A,B}$ par rapport à $[0, 1[$. Ainsi,

$$\forall r \in [0, 1[, \Xi_{A,B}(r) \geq 0 \text{ ssi } \frac{|B - A| + |B|}{-A(2B - A)} \leq 0 \text{ ou } \frac{|B - A| - |B|}{-A(2B - A)} \geq 1.$$

La condition $\frac{|B-A|+|B|}{-A(2B-A)} \leq 0$ impose $A = B = 0$, ce qui est impossible puisque $A(2B-A) < 0$. Nous étudions alors la condition $\frac{|B-A|-|B|}{-A(2B-A)} \geq 1$. Si $A < 0$ et $2B - A > 0$ (resp. $A > 0$ et $2B - A < 0$), la condition est vérifiée si et seulement si $B < 0$ (resp. $B \geq 0$) et est vérifiée si et seulement si $B \leq \frac{A+1}{2}$ (resp. $B \geq \frac{A-1}{2}$) si $B \geq 0$ (resp. $B < 0$) ; ce qui clôt la preuve. □

6.2.3 Classes d'équivalences

À ce stade, nous savons que toute distribution d'observations de n'importe quel GUM linéaire gaussien sous contrainte (6.11) est stationnaire gaussienne avec une structure de covariance géométrique de la forme $A^{k-1}B$, avec $(A, B) \in \mathcal{S}$. Réciproquement, nous nous demandons maintenant si pour n'importe quelle distribution de cette forme il existe un GUM capable de la modéliser. En d'autres termes, nous étudions l'image réciproque de la fonction

$$\varphi : (a, b, c, \eta) \mapsto (A = a + bc, B = ab^2\eta + bc). \tag{6.17}$$

Il est justement possible de calculer cette image réciproque ; nous avons la propriété suivante.

Proposition 1 *L'image réciproque φ^{-1} est l'application qui, pour tout $(A, B \in \mathcal{S})$, associe :*

- *si $A \neq B, B \neq 0$,*

$$\varphi^{-1}(A, B) = \left\{ \left(\frac{A - B}{1 - b^2\eta}, b, \frac{B - Ab^2\eta}{b(1 - b^2\eta)}, \eta \right) ; \eta > 0, b \in [-x_2, -x_1] \cup [x_1, x_2] \right\}, \tag{6.18}$$

où $x_j = \sqrt{\frac{1+2AB-A^2+(-1)^j\sqrt{(A^2-1)((2B-A)^2-1)}}{2\eta}}$ *pour* $j \in \{1,2\}$ *;*

- *si* $A = B, B \neq 0$,

$$\begin{aligned}\varphi^{-1}(A,A) = &\left\{\left(0, b, \frac{A}{b}, \eta\right); \eta > 0, b \in \left[-\frac{1}{\sqrt{\eta}}, -\frac{|A|}{\sqrt{\eta}}\right] \cup \left[\frac{|A|}{\sqrt{\eta}}, \frac{1}{\sqrt{\eta}}\right]\right\} \\ &\cup \left\{\left(a, \frac{1}{\sqrt{\eta}}, (A-a)\sqrt{\eta}, \eta\right); \eta > 0, a \in \mathbb{R}\right\} \\ &\cup \left\{\left(a, -\frac{1}{\sqrt{\eta}}, -(A-a)\sqrt{\eta}, \eta\right); \eta > 0, a \in \mathbb{R}\right\}; \end{aligned} \tag{6.19}$$

- *si* $A \in \mathbb{R}, B = 0$,

$$\begin{aligned}\varphi^{-1}(A,0) = &\left\{(a, 0, c, \eta); \eta > 0, a \in [-1,1], c \in \left[-\sqrt{(1-a^2)\eta}, \sqrt{(1-a^2)\eta}\right]\right\} \\ &\cup \left\{(a, b, -ab\eta, \eta); \eta > 0, a \in [-1,1], b \in \left[-\frac{1}{\sqrt{\eta}}, \frac{1}{\sqrt{\eta}}\right]\right\}. \end{aligned} \tag{6.20}$$

Preuve. Si $B \neq 0$, résoudre (6.17) est résumé par le système d'équations

$$(\mathfrak{S}) : \begin{cases} a + bc & = A \\ a(1 - b^2\eta) & = A - B \end{cases}. \tag{6.21}$$

Rappelons que les variances α et β sont nécessairement positives. Cela implique notamment que $b \in \left[-\frac{1}{\sqrt{\eta}}, \frac{1}{\sqrt{\eta}}\right]$. Soit $A \neq B, B \neq 0$. Puisque $A \neq B$, ni $a = 0$, ni $b = \pm\frac{1}{\sqrt{\eta}}$ pour tout $\eta > 0$, et $(\mathfrak{S})$ se résume à $bc = \frac{B - Ab^2\eta}{1-b^2\eta}$ et $a = \frac{A-B}{1-b^2\eta}$. Dans ce cas, b ne peut être égal à 0 et $c = \frac{B-Ab^2\eta}{b(1-b^2\eta)}$. Cependant, en notant $x = b^2\eta$, la condition « $\alpha \geq 0$ » donne

$$\Xi(x) = -x^2 + (1 + 2AB - A^2)x - B^2 \geq 0. \tag{6.22}$$

Le polynôme Ξ possède des racines réelles si et seulement si son discriminant est positif

$$\Delta = (A^2 - 1)((2B - A)^2 - 1) \geq 0; \tag{6.23}$$

ce qui est toujours vrai puisque $(A, B) \in \mathcal{S}$. Si $|A| \neq 1$, Ξ admet deux racines simples (ou une racine double) :

$$\forall j \in \{1, 2\}, x_j = \frac{1 + 2AB - A^2 + (-1)^j \sqrt{(A^2 - 1)((2B - A)^2 - 1)}}{2}. \tag{6.24}$$

Ces racines sont localisées dans $]0, 1[$; en effet, Ξ s'écrit $\Xi(x) = x(1-x)(1-A^2) - (B - Ax)^2$ et $-(B - Ax)^2$ est négatif pour tout $x \in \mathbb{R}$ tandis que la quantité $x(1 - x)(1 - A^2)$ est positive uniquement pour $x \in [0, 1]$. Si $A = \pm 1$, alors $\Xi(x) = -(B \mp x)^2$. La condition « $\alpha \geq 0$ » n'est atteinte qu'aux points $x = \pm B$ quand $B \in]-1, 0[$ si $A = -1$ ou $B \in]0, 1[$ si $A = 1$.

Prenons maintenant $A = B, B \neq 0 : A - B = 0$, alors soit $a = 0, c = \frac{A}{b}$ avec $\eta > 0$ et $b \in \left[-\frac{1}{\sqrt{\eta}}, \frac{1}{\sqrt{\eta}}\right] \setminus \{0\}$. La condition « $\alpha \geq 0$ » donne $b^2 \geq \frac{A^2}{\eta}$; soit $b = \pm\frac{1}{\sqrt{\eta}}$ avec $\eta > 0$, et alors $c = \pm(A - a)\sqrt{\eta}$ (sauf si $b = 0$ où c peut être choisi librement dans ce cas) et a peut être choisi librement.

Choisissons finalement $A \in \mathbb{R}, B = 0$. Alors, pour tout $A \in \mathbb{R}$ et pour tout $k \in \mathbb{N}^*$, $\mathrm{Cov}(x_t, x_{t+k}) = A^{k-1}B = 0$. Donc, soit $b = 0$, soit $ab\eta + c = 0$. Si $b = 0$, $(1 - a^2)\eta - c^2 \geq 0$ si et seulement si $a \in [-1, 1]$ et $c \in \left[-\sqrt{(1 - a^2)\eta}, \sqrt{(1 - a^2)\eta}\right]$. Sinon, $\eta - a^2\eta + a^2b^2\eta^2 - a^2b^2\eta^2 \geq 0$ si et seulement si $(1 - a^2) \geq 0$ si et seulement si $a \in [-1, 1]$. □

Grâce à la proposition 1, nous constatons d'abord que la fonction φ n'est pas injective puisque différents GUM peuvent mener à la modélisation d'une même distribution $p_{A,B}(x_{0:t})$. Cependant, cette fonction est surjective, c'est-à-dire pour tout $(A, B) \in \mathcal{S}$, il existe au moins une instance de GUM modélisant $p_{A,B}(x_{0:t})$. De plus, puisque la proposition 1 fournit une forme explicite de l'image réciproque, nous pouvons dorénavant étudier l'image réciproque φ^{-1} n'importe quel sous-modèle (en l'occurrence HMM et RNN) afin d'obtenir une cartographie de nos différents modèles.

6.2.4 Expressivité des RNN et HMM

Pour tout $(A, B) \in \mathcal{S}$ tel que $A \neq B, A \neq 0$ et $B \neq 0$ (nous omettons les cas aux bords qui n'introduisent pas de difficulté particulière), nous cherchons maintenant à dire si une distribution normale multivariée de moyenne $\mathbf{1}$ et de matrice de covariance $\boldsymbol{R}_t(A, B)$ peut être modélisée soit par un HMM, soit par un RNN, par les deux ou aucun. Pour cela, nous

évaluons l'ensemble des paramètres (a, b, c, η) qui

- appartiennent à l'image réciproque $\varphi^{-1}(A, B)$ (voir proposition 1),
- vérifient les contraintes propres aux modèles HMM, respectivement RNN.

Soit un paramètre $\theta = (a, b, c, \eta) \in \varphi^{-1}(A, B)$. Autrement dit, d'après la proposition 1 (équation (6.18)), les variables $\eta \in \mathbb{R}^{*+}$ et $b \in [-x_2, -x_1] \cup [x_1, x_2]$, ont été choisies arbitrairement, tandis que $a = \frac{A-B}{1-b^2\eta}$ et $c = \frac{B-Ab^2\eta}{b(1-b^2\eta)}$.

HMM Alors, θ décrit un HMM si et seulement si la contrainte $c = 0$ est vérifiée, c'est-à-dire si et seulement si

$$\frac{B - Ab^2\eta}{b(1-b^2\eta)} = 0 \overset{A\neq 0}{\Leftrightarrow} \frac{B}{A} = b^2\eta$$
$$\Leftrightarrow \frac{(1 + 2AB - A^2) - \sqrt{\Delta_{A,B}}}{2} \leq \frac{B}{A} \leq \frac{(1 + 2AB - A^2) + \sqrt{\Delta_{A,B}}}{2}$$

(où $\Delta_{A,B} = (A^2 - 1)((2B - A)^2 - 1)$ pour plus de lisibilité)

$$\Leftrightarrow 0 \leq \left(\frac{2B}{A} - (1 + 2AB - A^2)\right)^2 \leq (A^2 - 1)((2B - A)^2 - 1)$$
$$\Leftrightarrow 0 \leq (2B - A)^2(1 - A)^2 \leq A^2(1 - A^2)(1 - (2B - A)^2)$$
$$\Leftrightarrow 0 \leq (2B - A)^2 \leq A^2.$$

Cette dernière équivalence est permise car $(A, B) \in \mathcal{S}$. Avec l'hypothèse $B \neq 0$, nous pouvons alors conclure : θ décrit un HMM si et seulement si

$$AB \geq 0 \text{ et } |B| \leq |A|. \tag{6.25}$$

Cette conclusion ne change pas aux bords.

RNN Par ailleurs, θ décrit un RNN si et seulement si les contraintes $\alpha = 0$ (voir équation (6.14) et $\eta = c^2$ sont vérifiées. Plus particulièrement, nous obtenons le système à deux équations suivant :

$$
\begin{aligned}
&\begin{cases}
\left[1-\left(\frac{A-B}{1-b^2\eta}\right)^2-2\frac{(A-B)(B-Ab^2\eta)}{(1-b^2\eta)^2}\right]\eta-\left(\frac{B-Ab^2\eta}{b(1-b^2\eta)}\right)^2 &= 0\\
\eta &= \left(\frac{B-Ab^2\eta}{b(1-b^2\eta)}\right)^2
\end{cases}\\
\overset{A-B\neq 0}{\Leftrightarrow}&\begin{cases}
A-B+2(B-Ab^2\eta) &= 0\\
(B-Ab^2\eta)^2 &= b^2\eta(1-b^2\eta)^2
\end{cases}\\
\overset{A\neq 0}{\Leftrightarrow}&\begin{cases}
\frac{B}{A} &= 2b^2\eta-1\\
(B-Ab^2\eta)^2 &= b^2\eta(1-b^2\eta)^2
\end{cases}\\
\Leftrightarrow&\begin{cases}
\frac{B}{A} &= 2b^2\eta-1\\
A^2 &= b^2\eta
\end{cases}\\
\Leftrightarrow&\begin{cases}
B &= A(2A^2-1)\\
A^2 &= b^2\eta
\end{cases}.
\end{aligned}
$$

En étudiant les conditions aux bords dans le cadre du RNN, nous trouvons finalement que la fonction φ est surjective sur l'ensemble

$$
\begin{aligned}
\{(A,B)\in\mathcal{S};&(B=A(2A^2-1)\wedge A\in[-1,1])\\
&\vee(A=B\wedge A\in[-1,1])\\
&\vee(A\in\mathbb{R},B=0)\}.
\end{aligned}
\tag{6.26}
$$

GUM Déterministe (DGUM) Plus généralement (sans parler de RNN), nous nous sommes intéressés à l'apport de transitions probabilistes par rapport à des transitions déterministes. Nous avons considéré une troisième classe de modèles dans lequel nous imposons seulement $\alpha = 0$ (la contrainte $c^2 = \eta$ est relâchée par rapport à un RNN). Cela revient ainsi à considérer un RNN dont le vecteur latent n'est pas initialisé à partir d'une première observation et d'un vecteur latent nul, mais plutôt obtenu aléatoirement. À partir de cette seule contrainte, il est possible de vérifier que pour tout $A, B \in \mathcal{S}$, $\varphi^{-1}(A, B)$ n'est pas vide. Autrement dit, toute gaussienne $p_{A,B}(x_{0:t})$ avec une covariance géométrique peut être produite par un DGUM.

La figure 6.3 propose une cartographie dans le plan (A, B) des points observés ci-dessus. Notamment, nous pouvons constater qu'une distribution peut être modélisée par un HMM mais pas un RNN, l'inverse, les deux ou aucun. De plus, un modèle RNN ne dépend que

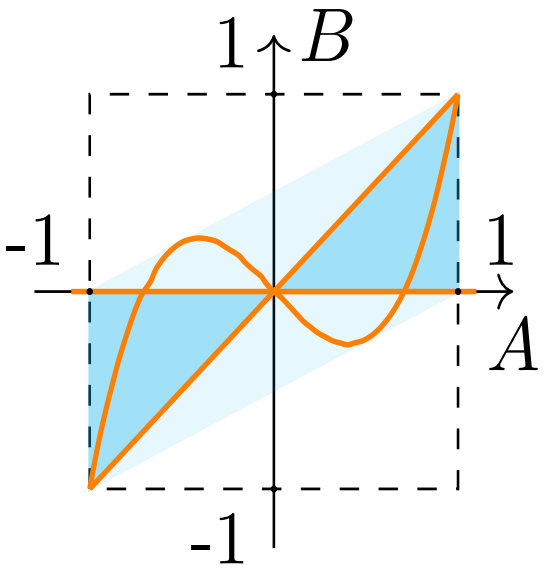


FIGURE 6.3 – Expressivité des RNN, HMM et GUM en fonction de A et B. La zone bleu pâle délimite $\mathcal{S}$: chaque point de cette zone correspond à une distribution de probabilités. Toute distribution de $\mathcal{S}$ peut être modélisée par au moins une instance de GUM et de DGUM. De plus, chaque distribution dans la zone bleue plus foncée (resp. la courbe orange) peut être modélisée par au moins une instance de HMM (resp. RNN).

de deux paramètres : η et le produit bc (ce produit vient de la contrainte d'initialisation $c^2 = \eta$), d'où la courbe. À l'inverse, un HMM dépend de trois paramètres a, b et η, d'où les deux triangles. Finalement, relâcher la contrainte d'initialisation dans le cas du RNN mène au DGUM dans lequel il ne reste plus que la contrainte $\alpha = 0$. Cette contrainte induit une relation entre les paramètres (à l'aide de (6.14), b peut être exprimé en fonction de a, c et η). Bien qu'un DGUM ait le même nombre de paramètres libres qu'un HMM, son pouvoir expressif surpasse celui du HMM puisque toute distribution de $\mathcal{S}$ peut être obtenue par un DGUM mais pas nécessairement par un HMM. Toutefois, cette conclusion est à nuancer par le caractère linéaire gaussien de notre étude.

6.3 Généralisation au cas multidimensionnel

Nous abordons désormais la question de l'extension de la section 6.2 au cas multidimensionnel. En effet, nous souhaitons lever l'hypothèse que h_t est de dimension 1, et étudier

l'apport de la dimension n du vecteur d'état h_t sur l'expressivité des modèles GUM, HMM et RNN (x_t reste ici scalaire).

Dans ce cadre, le modèle GUM est toujours défini par les équations (6.8)-(6.10) :

$$\begin{aligned} p(h_0) &= \mathcal{N}(h_0; 0; \eta), \\ p(h_t|h_{t-1}, x_{t-1}) &= \mathcal{N}(h_t; ah_{t-1} + cx_{t-1}; \alpha), \\ p(x_t|h_t) &= \mathcal{N}(x_t; bh_t; \beta), \end{aligned}$$

à cela près que les paramètres du modèle sont désormais des vecteurs ou des matrices.

Notons η_t la matrice de covariance de h_t (nous verrons plus tard dans quelle mesure η_t dépend de η et du temps t). On exprime alors dans le cas multidimensionnel la variance des observations à tout instant $t \in \mathbb{N}$ comme

$$\mathrm{Var}(x_t) = \beta + b\eta_t b^T, \tag{6.27}$$

ainsi que la fonction de covariance des observations :

$$\forall t \in \mathbb{N}, \forall k \in \mathbb{N}^*,\ \mathrm{Cov}(x_t, x_{t+k}) = \underbrace{b}_{H}(\underbrace{a + cb}_{F})^{k-1}(\underbrace{a\eta_t b^T + c(\beta + b\eta_t b^T)}_{N_t}) \tag{6.28}$$

qui généralise (6.16). Les équations (6.27) et (6.28) s'obtiennent de la même façon que dans le cas scalaire.

On observe donc d'emblée certaines difficultés par rapport au cas scalaire : la covariance $\mathrm{Cov}(x_t, x_{t+k})$ dépend *a priori* du temps et s'exprime en fonction de trois paramètres (H, F, N_t) (car on ne peut plus commuter la matrice centrale F^{k-1} avec le vecteur colonne N_t) ; enfin, la factorisation de la covariance (6.28) n'est pas unique pour des raisons que nous explicitons ci-dessous.

6.3.1 Stationnarité

Dans le cas unidimensionnel, nos contraintes et hypothèses menaient naturellement au fait que la variance de h_t était constante et égale à η. Dans le cas multidimensionnel, la

suite $(\eta_t)_{t\in\mathbb{N}}$ est définie par :

$$\begin{cases} \eta_0 & = \eta \\ \forall t \in \mathbb{N}, \eta_{t+1} & = (\alpha + c\beta c^T) + (a+cb)\eta_t(a+cb)^T \end{cases}. \tag{6.29}$$

Preuve. Soit $t \in \mathbb{N}$. Calculons la loi marginale des états cachés :

$$\begin{aligned} p(h_{t+1}) &= \iint p(h_{t+1}|h_t, x_t)p(x_t|h_t)p(h_t)dh_t dx_t \\ &= \iint \mathcal{N}(h_{t+1}; ah_t + cx_t; \alpha)\mathcal{N}(x_t; bh_t; \beta)\mathcal{N}(h_t; 0; \eta_t)dh_t dx_t \\ &= \int \mathcal{N}(h_t; 0; \eta_t)\mathcal{N}(h_{t+1}; (a+cb)h_t; \alpha + c\beta c^T)dh_t \\ &= \mathcal{N}(h_{t+1}; 0; \underbrace{\alpha + c\beta c^T + (a+cb)\eta_t(a+cb)^T}_{\eta_{t+1}}). \end{aligned}$$

□

L'équation (6.29) implique que

$$\forall t \in \mathbb{N}, \eta_{t+1} - \eta_t = (a+cb)^t \left[\eta_1 - \eta_0\right](a+cb)^{tT}. \tag{6.30}$$

Ainsi, d'après l'équation précédente, la suite (η_t) est constante si

$$\eta_0 = \eta_1 = \eta. \tag{6.31}$$

L'hypothèse (6.31) entraîne à son tour (d'après les équations (6.27) et (6.28)) que $\mathrm{Var}(x_t)$ et $\mathrm{Cov}(x_t, x_{t+k})$ ne dépendent plus du temps et donc que $(x_t)_{t\in\mathbb{N}}$ est un processus stationnaire au sens large. Notons enfin que, sous l'hypothèse (6.31), l'équation (6.29) devient :

$$\eta = (\alpha + c\beta c^T) + (a+cb)\eta(a+cb)^T. \tag{6.32}$$

Cette équation en η n'a un sens (rappelons que η est une matrice de covariance) que si elle admet une solution semi-définie positive, ce qui implique [Gevers and Wouters, 1978, Brockett, 2015] que

$$(a+cb) \text{ a toutes ses valeurs propres dans le disque unité ouvert } \{z \in \mathbb{C}; |z| < 1\}. \tag{6.33}$$

La contrainte (6.33) correspond à la contrainte $|A| < 1$ du cas scalaire. Nous supposerons désormais que (6.33) et (6.31) sont vérifiées, et donc que $x_{0:t}$ est stationnaire. Notons que cette hypothèse de stationnarité est raisonnable, car même sous la seule hypothèse (6.33), la suite (η_t) converge lorsque t tend vers l'infini. Par conséquent, $(x_t)_{t\in\mathbb{N}}$ est au minimum asymptotiquement stationnaire. Ainsi, $(h_t)_{t\in\mathbb{N}}$ comme $(x_t)_{t\in\mathbb{N}}$ sont asymptotiquement stationnaires.

6.3.2 Image réciproque de la fonction de covariance et théorie de la réalisation stochastique

Posons $r_k = \mathrm{Cov}(x_t, x_{t+k})$. Nous observons que cette fonction de covariance possède une structure très spécifique :

$$r_0 = \beta + b\eta b^T, \tag{6.34}$$

$$\forall k \in \mathbb{N}^*, r_k = \underbrace{b}_{H}\underbrace{(a+cb)}_{F}^{k-1}\underbrace{(a\eta b^T + c(\beta + b\eta b^T))}_{N}. \tag{6.35}$$

Nous pouvons voir (6.35) comme une fonction $(a, b, c, \alpha, \beta, \eta) \mapsto (r_k)_{k\in\mathbb{N}^*}$, dont il faudrait pouvoir comparer les espaces images obtenus sous les différents cas : celui du GUM, celui du HMM ($c = 0$) et celui du RNN ($\alpha = 0$), d'une part l'un par rapport à l'autre, et d'autre part par rapport à l'ensemble de toutes les suites $(r_k)_{k\in\mathbb{N}}$ qui sont des fonctions de covariance. Une telle étude semble difficile à appréhender ; pouvons-nous alors étudier l'image réciproque de l'application (6.35) ? Ce point de vue pose plusieurs questions :

(i) Quelles sont les conditions pour qu'une fonction de covariance $(r_k)_{k\in\mathbb{N}^*}$ se factorise sous la forme

$$r_k = HF^{k-1}N? \tag{6.36}$$

(ii) Soit $(r_k)_{k\in\mathbb{N}}$ une suite factorisable (c'est-à-dire vérifiant (6.36) pour tout $k \in \mathbb{N}^*$) ; sous quelles conditions $(r_k)_{k\in\mathbb{N}}$ est-elle une fonction de covariance ?

(iii) Soit $(r_k)_{k\in\mathbb{N}}$ une fonction de covariance factorisable ; sous quelles conditions cette fonction peut-elle être produite par un GUM ? un HMM ? un RNN ?

Une telle étude fait appel à la théorie de la réalisation stochastique, brièvement rappelée dans l'annexe A. En effet, rappelons que la fonction de covariance d'un GUM vérifie (6.35) et est donc pour tout $k \in \mathbb{N}^*$ de la forme

$$r_k = HF^{k-1}(FPH^T + S), \tag{6.37}$$

où P est une matrice vérifiant

$$P - FPF^T = Q. \tag{6.38}$$

La suite $(r_k)_{k\in\mathbb{N}}$ correspond à la covariance du système d'état décrit par

$$h_{t+1} = Fh_t + u_t \tag{6.39}$$

$$x_t = Hh_t + v_t \tag{6.40}$$

où h_0 est centré et décorrélé de (u_t, v_t), et où (u_t, v_t) est une variable aléatoire centrée, décorrélée, stationnaire au sens large et de matrice de covariance

$$\mathbb{E}\left[\begin{bmatrix} u_t \\ v_t \end{bmatrix} .[u_{t'}, v_{t'}]\right] = \begin{bmatrix} Q & S \\ S^T & R \end{bmatrix} \delta_{t'}(t). \tag{6.41}$$

Réciproquement, étant donné une fonction de covariance de la forme $HF^{k-1}N$, la théorie de la réalisation stochastique a pour vocation de la *réaliser* sous la forme d'un système d'état, c'est-à-dire de trouver au moins un système d'état (6.39)-(6.41), avec F de dimension minimale, tel que $r_k = HF^{k-1}N$ pour tout $k \in \mathbb{N}^*$. Cette réalisation de $(r_k)_{k\in\mathbb{N}}$ se déroule en deux étapes distinctes (voir annexe A) :

1. à partir de $(r_k)_{k\in\mathbb{N}^*}$, on trouve une matrice F (de dimension minimale) et deux vecteurs H^T et N tels que $r_k = HF^{k-1}N$;
2. connaissant r_0 et un triplet (H, F, N), on identifie les paramètres restants P, Q, R et S.

Il se trouve que ces deux étapes (réalisation déterministe puis réalisation stochastique) apportent des réponses aux questions (i) et (ii) posées plus haut.

Étape 1 et question (i)

Un rôle essentiel dans cette théorie est joué par les matrices de Hankel (c'est-à-dire dont tous les éléments d'une anti-diagonale sont égaux ; voir annexe A) de dimension infinie

$$\mathcal{R}_\infty = \begin{bmatrix} r_1 & r_2 & r_3 & \dots \\ r_2 & r_3 & & \\ r_3 & & & \\ \vdots & & & \end{bmatrix}. \tag{6.42}$$

En effet, si $r_k = HF^{k-1}N$, $\mathcal{R}_\infty$ se factorise en

$$\mathcal{R}_\infty = \begin{bmatrix} H \\ HF \\ HF^2 \\ \vdots \end{bmatrix} .[N, FN, F^2N, ...]. \tag{6.43}$$

De plus, $\mathcal{R}_\infty$ est de rang fini n (la dimension de F) si et seulement si chaque facteur est lui-même de rang complet n. Par ailleurs, en introduisant entre ces deux facteurs la matrice $I = TT^{-1}$ (où T est une matrice inversible quelconque), on obtient une factorisation équivalente. Réciproquement, si $\mathcal{R}_\infty$ se factorise en un produit de deux facteurs de dimension $(\infty \times n)$ et $(n \times \infty)$, tous deux étant de rang complet n, alors n est la dimension minimale de toute matrice F réalisant $r_k = HF^{k-1}N$ et toutes les solutions sont isomorphes. Plus précisément (voir [Ho and Kalman, 1966, proposition 3]), (H_1, F_1, N_1) et (H_2, F_2, N_2) sont deux réalisations minimales de $(r_k)_{k\in\mathbb{N}^*}$ si et seulement s'il existe T inversible telle que

$$\begin{cases} F_2 &= TF_1T^{-1}, \\ N_2 &= TN_1, \\ H_2 &= H_1T^{-1}. \end{cases} \tag{6.44}$$

Étape 2 et question (ii)

Pour aborder le problème (ii), une piste serait d'utiliser le théorème de Carathéodory-Toeplitz comme nous l'avons fait en section 6.2. Toutefois, ce raisonnement ne se généralise pas aisément dans le cas multidimensionnel, dans lequel nous sommes confrontés à des difficultés calculatoires. Ainsi, plutôt que de répondre au problème (ii) de manière explicite, nous y répondons implicitement grâce au lemme positif réel, qui se traduit par une condition d'existence d'une solution au problème de réalisation stochastique.

Lemme positif réel (voir annexe A)

À ce stade, l'étape 1 a permis d'identifier (à une matrice inversible T près) les paramètres (H, F, N) permettant de représenter $(r_k)_{k\in\mathbb{N}^*}$ sous la forme $r_k = HF^{k-1}N$. Rappelons cependant que $(r_k)_{k\in\mathbb{N}^*}$ est plus précisément de la forme (6.37), et que P, Q, R et S restent inconnues.

Cependant l'équation (6.37) provient du système (6.39)-(6.41). Or, (6.39)-(6.41) implique que

$$\begin{bmatrix} P & N \\ N^T & r_0 \end{bmatrix} - \begin{bmatrix} F \\ H \end{bmatrix} P \begin{bmatrix} F^T & H^T \end{bmatrix} = \begin{bmatrix} Q & S \\ S^T & R \end{bmatrix}, \tag{6.45}$$

$$P \geq 0, \tag{6.46}$$

$$\begin{bmatrix} Q & S \\ S^T & R \end{bmatrix} \geq 0. \tag{6.47}$$

Les contraintes (6.46) et (6.47) proviennent de ce que P et $\begin{bmatrix} Q & S \\ S^T & R \end{bmatrix}$ sont des matrices de covariance, donc semi-définies positives. Quant à l'équation (6.45), elle peut être vue comme un système de trois équations à quatre inconnues (P, Q, R et S), ou plutôt comme un système de trois équations à trois inconnues (Q, R et S) paramétrées par P. Finalement, P est le

paramètre des solutions du système contraint (6.45)-(6.47). Désignons par $\mathcal{P}$ l'ensemble des paramètres

$$\mathcal{P} = \{P \text{ tel que (6.45)-(6.47) sont vérifiées}\}. \tag{6.48}$$

Le système (6.45)-(6.47) décrit implicitement les solutions du problème inverse (réaliser une covariance sous forme d'un système d'état (6.39)-(6.41)), mais permet également de répondre à la question (ii) ci-dessus. En effet, si (6.39)-(6.41) existe, alors l'état de ce système admet une covariance. Ainsi, (6.46) et (6.47) sont vérifiées par construction et la fonction de covariance $(r_k)_{k\in\mathbb{N}}$ du processus $(x_t)_{t\in\mathbb{N}}$ est de la forme (6.37). Elle est alors positive par construction.

La réciproque est plus délicate. Partant d'une suite $(r_k)_{k\in\mathbb{N}}$ factorisable (c'est-à-dire de la forme $r_k = HF^{k-1}N$ pour tout $k \in \mathbb{N}^*$), on montre [Faurre, 1976] que cette suite est positive (et donc définit une fonction de covariance) s'il existe des matrices P, Q, R et S satisfaisant (6.45)-(6.47). Comme l'ensemble des solutions de ce système est paramétré par P, on peut énoncer le résultat suivant, lié à l'existence ou non d'un élément dans $\mathcal{P}$:

Lemme 6 (Lemme positif réel) *La suite $(r_k)_{k\in\mathbb{N}}$ est une fonction de covariance si et seulement si $\mathcal{P}$ est non vide.*

Notons enfin qu'il existe des algorithmes efficaces de construction d'éléments de $\mathcal{P}$ (voir [Faurre, 1979, Caines, 2018]).

6.3.3 Expressivité des GUM, HMM et RNN

Résumons le paragraphe 6.3.2.

Partant d'une suite quelconque $(r_k)_{k\in\mathbb{N}}$, cette suite est factorisable (c'est-à-dire qu'il existe un triplet (H, F, N) tel que $r_k = HF^{k-1}N$ pour tout $k \in \mathbb{N}^*$) si la matrice de Hankel $\mathcal{R}_\infty$ est de rang fini n. Ce rang n est par ailleurs la dimension minimale de toute réalisation de $(r_k)_{k\in\mathbb{N}}$.

Partant d'une suite $(r_k)_{k\in\mathbb{N}}$ factorisable, cette suite est une fonction de covariance si et seulement si $\mathcal{P} \neq \emptyset$ ($\mathcal{P}$ étant l'ensemble des solutions du système contraint (6.45)-(6.47))

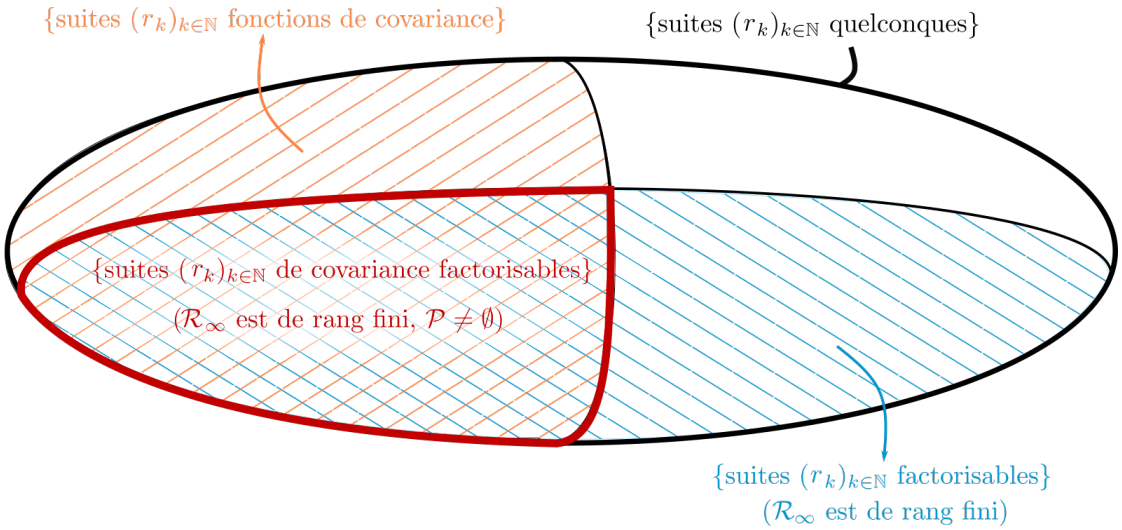


FIGURE 6.4 – Cette figure représente l'ensemble de toutes les suites réelles. En particulier, nous nous intéressons aux suites $(r_k)_{k\in\mathbb{N}}$ qui sont à la fois des fonctions de covariance et factorisables sous la forme $r_k = HF^{k-1}N$ (quart sud-ouest de la figure, hachures orange et bleues). Le calcul de la matrice $\mathcal{R}_\infty$ permet de passer de l'ensemble total à la moitié sud, tandis que le lemme positif réel permet de passer de la moitié sud au quart sud-ouest.

L'ensemble des suites $(r_k)_{k\in\mathbb{N}}$ qui sont à la fois des fonctions de covariance et qui sont factorisables (partie sud-ouest de la figure 6.4) est l'équivalent du parallélogramme de la figure 6.3. De même que dans le cas scalaire (voir section 6.2.4), nous allons maintenant étudier si tout point de cet ensemble est réalisable par un GUM, un HMM et/ou un RNN.

Expressivité des GUM

Cette question ne pose pas de difficulté particulière. En effet tout GUM peut s'écrire sous la forme

$$\begin{cases} h_{t+1} &= ah_t + cx_t + u'_t \\ x_t &= bh_t + v'_t \end{cases}, \tag{6.49}$$

$$\mathbb{E}\left[\begin{bmatrix} u'_t \\ v'_t \end{bmatrix} \cdot \begin{bmatrix} u_t'^T & v'_t \end{bmatrix}\right] = \begin{bmatrix} \alpha & 0 \\ 0 & \beta \end{bmatrix}. \tag{6.50}$$

En injectant la seconde équation de (6.49) dans la première, le système se réécrit comme

$$\begin{cases} h_{t+1} &= (a+cb)h_t + \underbrace{u'_t + cv'_t}_{u_t} \\ x_t &= bh_t + \underbrace{v't}_{v_t} \end{cases}, \tag{6.51}$$

$$\mathbb{E}\left[\begin{bmatrix} u_t \\ v_t \end{bmatrix} \cdot \begin{bmatrix} u_t^T & v_t \end{bmatrix}\right] = \begin{bmatrix} \alpha + c\beta c^T & c\beta \\ \beta c^T & \beta \end{bmatrix}. \tag{6.52}$$

On en déduit que tout GUM est un système d'état (6.39)-(6.41) de paramètres

$$F = a + cb, \quad H = b, \quad Q = \alpha + c\beta c^T, \quad R = \beta \text{ et } \quad S = c\beta. \tag{6.53}$$

Réciproquement, partant d'un système d'état (6.39)-(6.41), on peut identifier (si $R \neq 0$)

$$\alpha = Q - SR^{-1}S^T, \quad \beta = R, \quad b = H, \quad c = SR^{-1} \text{ et } \quad a = F - SR^{-1}H. \tag{6.54}$$

Notons que

$$\begin{bmatrix} \alpha & 0 \\ 0 & \beta \end{bmatrix} = \begin{bmatrix} I & -SR^{-1} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} Q & S \\ S^T & R \end{bmatrix} \begin{bmatrix} I & 0 \\ -R^{-1}S^T & 1 \end{bmatrix}, \tag{6.55}$$

ce qui assure que $\begin{bmatrix} \alpha & 0 \\ 0 & \beta \end{bmatrix}$ est semi-définie positive et donc que α l'est aussi.

Expressivité des HMM

Un modèle HMM, tel que nous l'avons défini en section 6.1, est un modèle (6.49)-(6.50) avec la contrainte supplémentaire $c = 0$. Il vérifie le modèle classique de chaîne de Markov cachée (dans tout ce chapitre, rappelons que nous sommes dans le cas gaussien et que les

covariances caractérisent les lois) dans le sens où

$$p(h_{0:t}, x_{0:t}) = \underbrace{p(h_0) \prod_{s=1}^{t} p(h_s|h_{s-1})}_{p(h_{0:t})} \underbrace{\prod_{s=0}^{t} p(x_s|h_s)}_{p(x_{0:t}|h_{0:t})} . \tag{6.56}$$

Ainsi, $h_{0:t}$ est une chaîne de Markov et n'est connue qu'au travers de l'observation $x_{0:t}$, d'où le nom de chaîne de Markov cachée. De plus, dans le modèle (6.56), $p(x_{0:t}|h_{0:t})$ est particulièrement simple, ce qui est dû à la nature bloc-diagonale de (6.50). Le modèle GUM (6.51)-(6.52) mérite tout autant le vocable de chaîne de Markov cachée puisque $h_{0:t}$ est markovien et n'est connu qu'au travers de $x_{0:t}$. La différence est la matrice de covariance (6.52), ce qui fait apparaître une corrélation entre u_t et v_t et induit une complexification de la loi conditionnelle $p(x_{0:t}|h_{0:t})$. Pour éviter toute confusion, nous conservons cependant le terme de chaîne de Markov cachée pour (6.49)-(6.50) avec $c = 0$, et de GUM pour (6.51)-(6.52).

Nous savons que toute fonction de covariance factorisable est réalisable par un GUM de même dimension. Partant d'une telle fonction, sous quelles conditions existe-t-il un HMM qui produit la même fonction de covariance ?

Supposons que $\dim(\mathcal{R}_\infty) = n$ et donc que $(r_k)_{k\in\mathbb{N}}$ admet une réalisation de dimension minimale n. Par souci de simplification, étudions si cette suite $(r_k)_{k\in\mathbb{N}}$ peut également être réalisée par un HMM *de même degré* n. Le problème revient donc à étudier si parmi l'ensemble des solutions, il en existe au moins une telle que $c = 0$. Cette question entraîne d'emblée la question suivante : bien que l'ensemble $\mathcal{P}_1$ dans lequel nous cherchons une solution HMM soit construit à partir *d'un* triplet (H_1, F_1, N_1), produit par l'étape de réalisation déterministe, ce triplet n'est pas unique ; si $\mathcal{P}_1$ ne devait pas contenir de solution HMM, est-ce qu'un autre ensemble $\mathcal{P}_2$, construit à partir d'un autre triplet (H_2, F_2, N_2), pourrait néanmoins contenir une solution HMM ? Il nous faut donc étudier la relation entre $\mathcal{P}_1$ et $\mathcal{P}_2$.

Rappelons que pour $(r_k)_{k\in\mathbb{N}}$ donnée, l'étape de réalisation déterministe produit un triplet (H_1, F_1, N_1) de degré minimal. Toute autre solution

$$(H_2, F_2, N_2) = (H_1 T_{12}^{-1}, T_{12} F_1 T_{12}^{-1}, T_{12} N_1) \tag{6.57}$$

s'obtient à partir de (H_1, F_1, N_1) par l'action d'une matrice inversible T_{12} quelconque. La

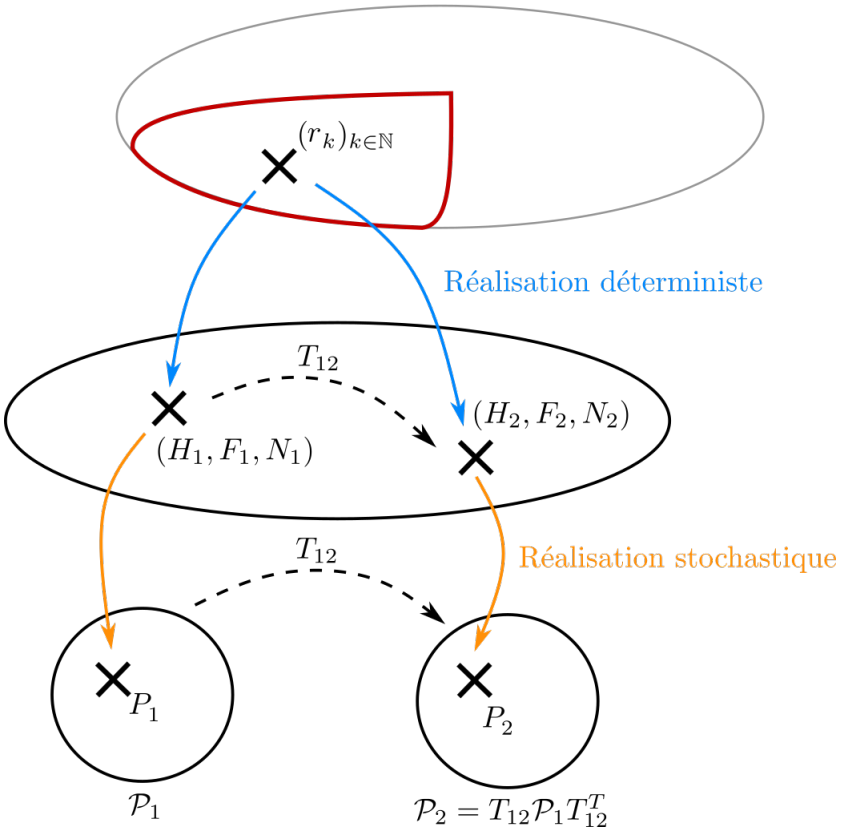


FIGURE 6.5 – À partir d'une fonction de covariance factorisable $(r_k)_{k\in\mathbb{N}}$ (voir figure 6.4), l'étape de réalisation déterministe (en bleu) consiste à trouver un triplet (H, F, N) représentant la fonction étudiée. Cette étape fournit une solution mais il existe une infinité de solutions à ce problème. Ces solutions sont isomorphes et il suffit de connaître la matrice inversible T_{12} appropriée pour passer d'une solution donnée (H_1, F_1, N_1) à une autre (H_2, F_2, N_2) (flèche pointillée).
L'étape de réalisation stochastique (en orange) revient à trouver un système d'état modélisant la fonction $(r_k)_{k\in\mathbb{N}}$ à partir du triplet (H, F, N) obtenu à l'étape précédente. Un triplet (H_i, F_i, N_i) mène à un ensemble de solutions $\mathcal{P}_i$. Ces ensembles sont également isomorphes et T_{12} suffit pour passer de $\mathcal{P}_1$ à $\mathcal{P}_2$, respectivement obtenus à partir de (H_1, F_1, N_1) et (H_2, F_2, N_2) (flèche pointillée).

deuxième étape, appliquée à (H_1, F_1, N_1) (respectivement (H_2, F_2, N_2)) fournit un ensemble de solutions paramétrées par $\mathcal{P}_1$ (respectivement $\mathcal{P}_2$).

Pour $i \in \{1, 2\}$, appelons « problème i » la recherche de solutions de l'équation (6.45)-(6.47), associée au triplet (H_i, F_i, N_i). Soit P_1 une solution du problème 1. En prémultipliant (respectivement postmultipliant) l'équation (6.45), de paramètres (H_1, F_1, N_1), par $\begin{bmatrix} T_{12} & 0 \\ 0 & 1 \end{bmatrix}$ (respectivement $\begin{bmatrix} T_{12} & 0 \\ 0 & 1 \end{bmatrix}^T$), on montre sans difficulté particulière que

$$\begin{aligned} \mathcal{P}_2 &= \{\text{solutions du problème 2}\} \\ &= \{T_{12} P_1 T_{12}^T \,;\, P_1 \in \mathcal{P}_1\}, \end{aligned}$$

ce que nous noterons globalement par l'équation ensembliste

$$\mathcal{P}_2 = T_{12} \mathcal{P}_1 T_{12}^T. \tag{6.58}$$

Soit maintenant un triplet (H_1, F_1, N_1) et une solution $P_1 \in \mathcal{P}_1$, telle que

$$N_1 = F_1 P_1 H_1^T. \tag{6.59}$$

Cette matrice P_1 est donc une solution HMM de $(r_k)_{k\in\mathbb{N}}$. Soit $(H_2, F_2, N_2) = (H_1 T_{12}^{-1}, T_{12} F_1 T_{12}^{-1}, T_{12} N_1)$, avec T_{12} une matrice inversible ; l'équation (6.59) est équivalente à

$$\underbrace{T_{12} N_1}_{N_2} = \underbrace{T_{12} F_1 T_{12}^{-1}}_{F_2} \underbrace{T_{12} P_1 T_{12}^T}_{P_2 \in \mathcal{P}_2} \underbrace{T_{12}^{-1T} H_1^T}_{H_2^T}, \tag{6.60}$$

donc $P_2 = T_{12} P_1 T_{12}^T$ est un représentant HMM appartenant à l'ensemble $\mathcal{P}_2$. Autrement dit, s'il existe un représentant HMM dans l'ensemble $\mathcal{P}_1$ associé à un représentant (H_1, F_1, N_1) de la classe d'équivalence produite par l'étape de réalisation déterministe, alors tout ensemble $\mathcal{P}_2 = T_{12} \mathcal{P}_1 T_{12}^T$ (avec T_{12} inversible quelconque) contient également une solution HMM. De même, s'il n'existe pas de tel représentant dans $\mathcal{P}_1$, alors aucun ensemble $\mathcal{P}_2 = T_{12} \mathcal{P}_1 T_{12}^T$ n'en contient non plus. Il suffit donc de rechercher une solution HMM dans l'ensemble $\mathcal{P}$ fourni par l'algorithme de réalisation stochastique, sans plus se soucier désormais des autres éléments de la classe d'équivalence. Nous avons obtenu le résultat suivant :

Proposition 2 *Soit $(r_k)_{k\in\mathbb{N}}$ une fonction de covariance factorisable et soit (H, F, N) un triplet (avec F de dimension minimale n) obtenu par réalisation déterministe. La suite $(r_k)_{k\in\mathbb{N}}$ peut être réalisée par un HMM de dimension n si et seulement s'il existe $\tilde{P}$ (et donc $\tilde{Q}(\tilde{P})$ et $\tilde{R}(\tilde{P})$) tel que*

$$\begin{bmatrix} \tilde{P} & N \\ N^T & r_0 \end{bmatrix} - \begin{bmatrix} F \\ H \end{bmatrix} \tilde{P} \begin{bmatrix} F^T & H^T \end{bmatrix} = \begin{bmatrix} \tilde{Q} & 0 \\ 0 & \tilde{R} \end{bmatrix}, \tag{6.61}$$

$$\tilde{P} > 0, \tag{6.62}$$

$$\begin{bmatrix} \tilde{Q} & 0 \\ 0 & \tilde{R} \end{bmatrix} \geq 0. \tag{6.63}$$

Soit $\tilde{\mathcal{P}}$ l'ensemble des solutions $\tilde{P}$ du problème contraint (6.61)-(6.63) On peut remarquer que $\tilde{\mathcal{P}}$ est un sous-ensemble convexe de $\mathcal{P}$. D'autre part, par rapport à $\mathcal{P}$, (6.61) entraîne la contrainte supplémentaire

$$N = F\tilde{P}H^T. \tag{6.64}$$

Cette équation ne peut être vérifiée que si $N \in \mathrm{Im}(F)$. Par ailleurs, si F est inversible, (6.64) implique également

$$H^T F^{-1} N > 0. \tag{6.65}$$

Donc si (6.64) et/ou (6.65) n'est pas vérifiée, alors la séquence $(r_k)_{k\in\mathbb{N}}$ ne peut pas être réalisée par un HMM de dimension n. La réciproque reste cependant à étudier.

Expressivité des RNN

De même que pour les HMM, l'étude peut se faire à partir de n'importe quel triplet (H, F, N) fourni par l'étape de réalisation déterministe. Dans (6.52), la condition $\alpha = 0$

implique que la matrice de covariance est égale à

$$\mathbb{E}\left[\begin{bmatrix} u_t \\ v_t \end{bmatrix} \cdot \begin{bmatrix} u_t & v_t \end{bmatrix}\right] = \begin{bmatrix} c \\ 1 \end{bmatrix} \beta \begin{bmatrix} c^T & 1 \end{bmatrix}, \tag{6.66}$$

et est donc de rang 1. La suite $(r_k)_{k\in\mathbb{N}}$ est donc réalisable par un RNN si et seulement s'il existe $\tilde{P} \in \mathcal{P}$ tel que

$$\begin{bmatrix} \tilde{P} & N \\ N^T & r_0 \end{bmatrix} - \begin{bmatrix} F \\ H \end{bmatrix} \tilde{P} \begin{bmatrix} F^T & H^T \end{bmatrix} = \begin{bmatrix} c \\ 1 \end{bmatrix} \beta \begin{bmatrix} c^T & 1 \end{bmatrix}, \tag{6.67}$$

$$\tilde{P} > 0, \tag{6.68}$$

pour un certain vecteur c et un certain $\beta > 0$.

6.3.4 Lien avec le cas unidimensionnel

Dans ce dernier paragraphe, nous nous proposons d'illustrer le paragraphe précédent en résolvant explicitement les équations dans le cas scalaire, puis nous ferons le lien avec les résultats obtenus en section 6.2.

Étape de réalisation stochastique

Considérons une fonction de covariance $(r_k)_{k\in\mathbb{N}}$ factorisable sous la forme $HF^{k-1}N$ pour tout $k \in \mathbb{N}^*$ avec H, F et N scalaires. Nous supposons que nous avons obtenu un tel triplet (H, F, N) par réalisation déterministe de $(r_k)_{k\in\mathbb{N}}$, et nous cherchons les paramètres scalaires P, Q, R, S vérifiant le système contraint (6.45)-(6.47), chacune de ces équations devenant

respectivement

$$\begin{cases} Q &= P(1-F^2) \\ R &= r_0 - PH^2 \\ S &= N - HFP \end{cases}, \tag{6.69}$$

$$P \geq 0, \tag{6.70}$$

$$\begin{cases} QR - S^2 &\geq 0 \\ Q &\geq 0 \\ R &\geq 0 \end{cases}. \tag{6.71}$$

En particulier, le système de contraintes (6.71) est la traduction de la contrainte de semi-définie positivité (6.47) lorsque Q, R et S sont scalaires.

Nous devons mettre en évidence l'ensemble $\mathcal{P}$ des nombres P positifs solutions de ce système. En utilisant (6.69), la première inégalité de (6.71) s'écrit

$$\Xi_{/H^2}(P) = -H^2P^2 + [r_0(1-F^2) + 2HFN]P - N^2 \geq 0. \tag{6.72}$$

Si l'on note

$$A = F \text{ et } B = HN, \tag{6.73}$$

alors le polynôme $\Xi_{/H^2}(P)$ correspond au polynôme $\Xi(x)$ présenté en (6.22) ; plus exactement,

$$H^2\, \Xi_{/H^2}(P) = \Xi(H^2P). \tag{6.74}$$

Nous déduisons alors les racines $P_{/H^2,i}$, avec $i \in \{1,2\}$, de $\Xi_{/H^2}$ à partir de celles de Ξ :

$$P_{/H^2,i} = \frac{(2HFN - r_0(1-F^2)) + (-1)^i\sqrt{(1-F^2)(r_0(1+F) - 2HN)(r_0(1-F) + 2HN)}}{2H^2}. \tag{6.75}$$

Puisque $\Xi_{/H^2}$ est concave, l'ensemble $\mathcal{P}$ des nombres P positifs solutions du système contraint (6.69)-(6.71) est inclus dans l'intervalle $[P_{/H^2,1}, P_{/H^2,2}]$. Par ailleurs, la contrainte (6.70) et la contrainte $R \geq 0$ de (6.71) imposent que P appartient à $[0, \frac{r_0}{H^2}]$; mais nous avons montré

que $[P_{/H^2,1}, P_{/H^2,2}] \subseteq [0, \frac{r_0}{H^2}]$ (voir calcul de la section 6.2), donc les contraintes $P \geq 0$ et $R \geq 0$ n'apportent pas de restrictions supplémentaires. La contrainte $Q \geq 0$ de (6.71) est quant à elle vérifiée du moment que $|F| \leq 1$. En définitive, nous obtenons dans le cas unidimensionnel une version explicite du théorème 7 de [Faurre, 1979] :

$$P_* = P_{/H^2,1}, P^* = P_{/H^2,2} \text{ et } \mathcal{P} = [P_*, P^*]. \tag{6.76}$$

On montre enfin aisément que $P_* > 0$ si $N > 0$ (le cas $P_* = 0$ n'est possible que si $\Xi_{/H^2}(0) = -N^2 \geq 0$, et donc $N = 0$, ce qui correspond au cas dégénéré d'une suite $(r_k)_{k\in\mathbb{N}}$ qui est nulle partout sauf en $k = 0$ où elle vaut r_0), donc $\mathcal{P}$ est un ensemble *défini* positif, ce qui est en concordance avec la théorie de Faurre (voir [Faurre, 1979, théorème 7]).

Application de l'isomorphisme, ensemble de toutes les solutions

Grâce à l'identification (6.54) des paramètres d'un GUM à partir de ceux d'un système d'état et au système (6.69), l'ensemble des GUM correspondant au triplet (H, F, N) est décrit par le système

$$\begin{cases} P & \in [P_{/H^2,1}, P_{/H^2,2}] \\ a & = \frac{Fr_0 - HN}{r_0 - H^2P} \overset{(6.73)}{=} \frac{Ar_0 - B}{r_0 - H^2P} \\ b & = H \\ c & = \frac{HN - FH^2P}{H(r_0 - H^2P)} \overset{(6.73)}{=} \frac{B - AH^2P}{H(r_0 - H^2P)} \end{cases}. \tag{6.77}$$

Ce système décrit l'ensemble des solutions paramétré par l'ensemble $\mathcal{P}$ décrit par (6.76), qui, lui-même, dépend du triplet (H, F, N) produit par l'étape de réalisation déterministe. Cherchons maintenant l'ensemble de toutes les solutions, lorsqu'on balaye l'ensemble de tous les triplets (H, F, N) possibles. Rappelons (voir l'équation (6.57) et la figure 6.5) que les triplets se déduisent les uns des autres par l'action d'une matrice inversible quelconque, c'est-à-dire ici, dans le cas scalaire, par l'action d'un nombre $T \in \mathbb{R}^*$ quelconque. On en déduit que l'ensemble de toutes les solutions (c'est-à-dire lorsque T parcourt $\mathbb{R}^*$) est décrit

par

$$\begin{cases} b & \in \mathbb{R}^* \\ P & \in [P_{/b^2,1}, P_{/b^2,2}] \\ a & = \frac{Ar_0 - B}{r_0 - b^2 P} \\ c & = \frac{B - Ab^2 P}{b(r_0 - b^2 P)} \end{cases} \Leftrightarrow \begin{cases} P & > 0 \\ b & \in [-x_2, -x_1] \cup [x_1, x_2] \\ a & = \frac{Ar_0 - B}{r_0 - b^2 P} \\ c & = \frac{B - Ab^2 P}{b(r_0 - b^2 P)} \end{cases}, \tag{6.78}$$

où x_1, x_2 sont donnés en (6.18). En conclusion, la proposition 1 de la section 6.2.3 (écrite dans le cas $r_0 = 1$) résout implicitement le problème de départ en deux étapes : calcul de l'ensemble $\mathcal{P}$ (puis de la solution (a, b, c) associée) à partir du triplet (H, F, N) produit par l'étape de réalisation déterministe, puis passage de cette solution particulière à l'ensemble de toutes les solutions.

6.4 Conclusion et perspectives

Dans ce chapitre, nous avons étudié l'expressivité (ou pouvoir modélisant) des modèles HMM et RNN, tous deux populaires pour l'analyse de données temporelles. La question est de savoir quelles sont les lois d'une séquence d'observations qui sont atteignables par chacune des deux structures, lorsqu'on en fait varier les paramètres dans les plages autorisées ; ces deux ensembles de lois sont-ils inclus l'un dans l'autre, disjoints, ou partagent-ils une intersection commune ? La question nous est venue du fait que les HMM et les RNN sont deux modèles probabilistes voisins, chacun des deux ayant une propriété qui (*a priori* tout au moins) semble l'appauvrir par rapport à l'autre : dans le cas des HMM, toutes les transitions sont probabilistes, mais il n'y a pas de lien direct entre l'état à l'instant t et l'observation à l'instant précédent ; dans le cas des RNN, l'état à l'instant t dépend de l'état et de l'observation à l'instant $t - 1$, mais de façon déterministe.

Pour répondre à cette question nous avons tout d'abord rassemblé les deux structures dans un cadre commun, en montrant que les HMM et les RNN, en tant que modèles probabilistes, pouvaient être vus comme deux cas particuliers d'un modèle génératif unifié que nous avons appelé GUM (pour *generative unified model*). Nous avons alors comparé l'expressivité des GUM, des HMM et des RNN. Pour que notre étude puisse être menée de façon analytique, nous nous sommes placés dans un premier temps dans le cas scalaire, linéaire et gaussien, avec (pour les RNN) une fonction d'activation linéaire. Sous ces conditions,

toutes les lois d'intérêt sont gaussiennes, et en particulier la loi d'une séquence d'observations est régie par la fonction de covariance du processus observé, qui se trouve avoir une structure géométrique particulière. Nous avons alors étudié, pour chaque structure, l'image réciproque de la fonction qui, à un jeu de paramètres donnés, associe la fonction de covariance de l'observation. Cette fonction est surjective, ce qui signifie que n'importe quelle fonction de covariance géométrique peut être produite par un GUM particulier. Par ailleurs, les fonctions de covariance atteintes par un HMM ou un RNN sont deux sous-ensembles de l'ensemble total dont aucun n'est inclus dans l'autre ; il existe des fonctions de covariance atteignables par un HMM mais pas par un RNN, ou réciproquement, ainsi que des fonctions de covariance atteignables par les deux modèles ou par aucun des deux. Enfin un RNN semble moins expressif qu'un HMM ; ce constat repose principalement sur la contrainte à l'initialisation d'un RNN que ne possède pas un HMM : sans cette contrainte, un RNN a l'expressivité d'un GUM, mais cela reste néanmoins à tempérer par le fait que les transitions dans notre étude sont linéaires et gaussiennes.

Nous avons ensuite essayé de lever l'hypothèse que l'état était de dimension 1. Le problème comporte alors des difficultés techniques nouvelles ; même si la fonction de covariance garde une structure factorisable remarquable, l'étude de l'image réciproque évoquée ci-dessus fait notamment appel à la théorie de la réalisation stochastique. Nous retrouvons dans le cas multidimensionnel la surjectivité des modèles GUM, et donnons un ensemble de conditions devant être vérifiées par une fonction de covariance pour qu'elle soit atteignable par un HMM et/ou un RNN de même dimension.

Finalement, cette étude apporte un certain nombre de réponses au problème étudié, mais soulève également quelques nouvelles questions, qui sont autant de sujets de futures recherches :

- *Forme explicite des solutions.* Nos caractérisations de solutions HMM ou RNN font appel au lemme positif réel et sont donc implicites ; il conviendrait de proposer des solutions analytiques, ce qui permettrait, partant d'une fonction de covariance factorisable, de pouvoir décider simplement si elle est réalisable par un HMM ou un RNN de même dimension ;
- *Rôle de la dimension de l'état.* Il serait bon d'apprécier de quelle façon la dimension de l'état joue sur l'expressivité des différents modèles ; en particulier, l'ensemble des fonctions de covariance factorisables produit par un GUM de dimension n augmente-t-il avec n ? par ailleurs peut-on atteindre n'importe quelle fonction de covariance

factorisable, produite par un GUM de dimension n, par un HMM et/ou un RNN, dès lors qu'on augmente leur dimension ?

- *Hypothèses moins restrictives.* Par ailleurs, il serait intéressant de lever la restriction que l'observation est de dimension 1 et, surtout, de prendre en compte dans notre analyse l'aspect non linéaire de la fonction d'activation d'un réseau de neurones. Dans ce but, une piste pour étendre nos résultats au second ordre serait d'utiliser la *unscented transform* (*transformée sans parfum*) [Julier et al., 2000, Julier and Uhlmann, 2004, Menegaz et al., 2015], qui est une méthode d'approximation des moments des deux premiers ordres d'une variable aléatoire après modification de celle-ci par une transformation non linéaire.

Conclusion et perspectives

Un réseau implique un grand nombre d'éléments et met en jeu des technologies variées, ce qui rend sa maintenance difficile. Par conséquent, il est crucial pour un administrateur de pouvoir identifier efficacement la cause d'une panne et de pouvoir anticiper le comportement d'un réseau pour un scénario de panne donné. Dans cette thèse, nous avons proposé et étudié des algorithmes et des modèles pour répondre à ces deux problèmes.

Dans un premier temps, nous avons proposé une structure de données appelée DIG-DAG dont le but est de mémoriser les liens de causalités entre les événements observés dans un log d'alarmes. Ces liens forment des motifs de causalité qui permettent de remonter à la cause racine d'une panne. Cette structure présente plusieurs avantages :

- la construction de cette structure peut être faite en ligne : elle peut être mise à jour en temps réel à la réception d'un nouvel événement dans le log ;
- le DIG-DAG permet de tenir compte d'informations connues *a priori* telles que la topologie du réseau ;
- le DIG-DAG permet de retrouver les motifs de causalités conformes aux requêtes de l'utilisateur ;
- par construction, le DIG-DAG est complètement interprétable.

Nous avons implémenté et testé notre solution sur plusieurs logs réels. Ces travaux ont suscité l'intérêt des *business units* de Nokia qui sont en train de l'intégrer dans des outils Nokia. Certaines extensions peuvent néanmoins être envisagées. Malgré les solutions proposées pour analyser efficacement des logs volumineux, le passage à l'échelle reste un défi. En effet, le DIG-DAG stocke toutes les chaînes de causalité observées dans le log au risque de mémoriser des motifs peu pertinents. Une piste d'amélioration consisterait donc à proposer un mécanisme pour les oublier. Par ailleurs, certaines parties d'un réseau peuvent avoir des comportements similaires. C'est par exemple le cas d'un réseau cellulaire composé

de plusieurs antennes identiques. Cela soulève deux problématiques : d'une part, identifier les comportements analogues, et d'autre part, les mutualiser dans le DIG-DAG. Enfin, le DIG-DAG peut être utilisé comme prédicteur. Nous avons défini des pondérations qui permettent l'analyse de causes racines. Par contre, ces pondérations ne sont pas adaptées pour la prédiction et une piste de recherche consisterait à trouver quelles pondérations conduiraient à une bonne prédiction. Un premier pas serait de chercher une version probabiliste du DIG-DAG. Ce serait alors l'occasion de faire le pont avec d'autres types de modèles tels que les modèles génératifs. À plus long terme, le DIG-DAG permet d'envisager des outils d'analyse automatique voire des outils de réparation automatique. En effet, le DIG-DAG permet d'associer une métadonnée arbitraire à chaque contexte de panne observé, et donc de déclencher une procédure d'alerte et/ou de réparation.

Dans un second temps, nous avons étudié deux modèles génératifs populaires pour l'analyse de données temporelles : les modèles de Markov cachés (HMM) et les réseaux de neurones récurrents (RNN). La loi des observations qu'ils modélisent permet, par exemple, de prédire le comportement d'une série temporelle à partir d'observations passées. Pour les comparer, nous avons proposé un modèle génératif unifié (GUM) afin de lier modèles de Markov cachés et réseaux de neurones dans un cadre théorique commun. En particulier, nous avons évalué leur expressivité dans un cadre simplifié : celui des GUM linéaires gaussiens. Nous avons notamment montré que l'ensemble des distributions génératives modélisables par un GUM linéaire gaussien est égal à l'ensemble des lois normales multivariées de fonction de covariance géométrique.

Dans le cas unidimensionnel, nous avons pu donner une formulation explicite des GUM correspondant à une fonction de covariance géométrique donnée, et nous avons pu instancier explicitement les cas particuliers propres aux HMM et aux RNN. Cela nous a permis de conclure dans le cas linéaire gaussien unidimensionnel qu'un HMM est plus expressif qu'un RNN. Cependant, il faut garder en tête que le choix du modèle le plus expressif n'est pas toujours le meilleur : en particulier, un modèle très expressif possédant un grand nombre de paramètres peut conduire à un surapprentissage au cours de son entraînement, en fonction du nombre de données d'entraînement et de leur nature.

Dans le cas multidimensionnel, l'ensemble des GUM correspondant à une fonction de covariance géométrique donnée n'est pas connu de manière analytique ; la théorie des systèmes nous indique qu'il en existe toujours au moins un (au travers d'un algorithme de construction), en revanche, elle ne permet pas de déterminer explicitement s'il existe ou

non un HMM ou un RNN de même dimension modélisant cette fonction de covariance. Il serait donc souhaitable, à terme, de proposer une cartographie comparée des HMM, des RNN et des GUM comme nous l'avions fait dans le cas unidimensionnel. Il reste également à comprendre le rôle de la dimension des paramètres d'un modèle donné sur son expressivité, et notamment de comparer l'expressivité de deux modèles GUM de dimensions différentes. Enfin, nous aimerions généraliser notre étude au cas non linéaire et/ou non gaussien, par exemple en considérant des fonctions d'activation linéaires par morceaux.

Pour conclure, cette thèse a exploré et croisé différents domaines de recherche afin de simplifier et d'automatiser la gestion de réseaux de télécommunication. Cependant, à l'heure où l'intelligence artificielle est en plein essor, il est important de continuer à maîtriser ce que de tels modèles peuvent apprendre et dans quel cadre les appliquer. Nous espérons que cette thèse s'inscrit dans cette optique et encouragera d'autres travaux à faire de même.

Annexe A

Introduction à la théorie de la réalisation

Dans cette annexe, nous présentons brièvement les éléments de la théorie de la réalisation déterministe et de la théorie de la réalisation stochastique dont nous avons besoin dans la section 6.3 [Faurre, 1976, Gevers and Wouters, 1978, Faurre, 1979, Gevers, 2006, Caines, 2018]. La réalisation stochastique est une branche de la théorie des systèmes : à la charnière du traitement du signal et de l'automatique, cette théorie s'intéresse à la représentation, au contrôle et à l'estimation de systèmes temporels dynamiques (voir par exemple [Chen, 1970, Kailath, 1980, Chui and Chen, 2012]).

A.1 Théorie de la réalisation déterministe

Considérons un système d'état linéaire et à temps discret, d'état interne h_t :

$$\begin{cases} h_{t+1} &= Fh_t + Nu_t \\ x_t &= Hh_t \end{cases}, \tag{A.1}$$

où F, N, et H sont des matrices de dimensions respectives $n \times n, n \times 1$, et $1 \times n$ (nous considérons ici uniquement le cas où l'observation x_t et l'entrée u_t sont de dimension 1). La

relation entre l'entrée u_t et la sortie x_t est décrite par l'équation de convolution

$$x_t = \sum_{k=1}^{+\infty} H_k u_{t-k}, \tag{A.2}$$

où les éléments H_k de la réponse impulsionnelle (encore appelés paramètres de Markov du système) sont donnés par l'équation

$$H_k = HF^{k-1}N, \forall k \geq 1. \tag{A.3}$$

De façon équivalente, la fonction de transfert strictement causale

$$H(z) = \sum_{k=1}^{+\infty} H_k z^{-k} \tag{A.4}$$

s'écrit sous la forme

$$H(z) = H(zI - F)^{-1}N. \tag{A.5}$$

Le problème de la réalisation déterministe consiste à construire trois matrices H, F, N, avec $F_{n\times n}$ de dimension minimale, à partir de la connaissance de la réponse impulsionnelle du système, c'est-à-dire à passer de la représentation infinie $(H_k)_{k\in\mathbb{N}^*}$ à une représentation finie (H, F, N), avec F de dimension minimale.

L'outil clé pour aborder ce problème est la matrice de Hankel [1] infinie

$$\mathcal{H}_\infty = \begin{bmatrix} H_1 & H_2 & H_3 & \dots \\ H_2 & H_3 & & \\ H_3 & & & \\ \vdots & & & \end{bmatrix}. \tag{A.6}$$

1. Pour rappel, une matrice de Hankel est une matrice dont tous les éléments d'une anti-diagonale sont égaux.

Du fait de (A.3), $\mathcal{H}_\infty$ se factorise en

$$\mathcal{H}_\infty = \begin{bmatrix} H \\ HF \\ HF^2 \\ \vdots \end{bmatrix} .[N, FN, F^2N, ...], \tag{A.7}$$

et est donc de rang fini. De plus, son rang est égal à n si chacun des deux facteurs de l'équation (A.7) est de rang complet. Réciproquement, si $\mathcal{H}_\infty$ est de rang fini, du fait de sa structure de Hankel, il existe $F_{n\times n}, N_{n\times 1}, H_{1\times n}$ telles que (A.3) est vérifiée. Par ailleurs, d'après la proposition ci-dessous, il se trouve que toutes les réalisations minimales de $H(z)$ sont isomorphes :

Proposition 3 (proposition 3 de [Ho and Kalman, 1966]) *(H_1, F_1, N_1) et (H_2, F_2, N_2) sont deux réalisations minimales de $H(z)$ si et seulement s'il existe T inversible telle que*

$$\begin{aligned} F_2 &= TF_1T^{-1}, \\ N_2 &= TN_1, \\ H_2 &= H_1T^{-1}. \end{aligned}$$

□

Notons enfin que des algorithmes de réalisation déterministe numériquement efficaces ont été proposés dans [de Jong, 1975, de Jong, 1978].

A.2 Théorie de la réalisation stochastique

Désormais, considérons le système d'état

$$\begin{cases} h_{t+1} &= Fh_t + u_t \\ x_t &= Hh_t + v_t \end{cases}, \tag{A.8}$$

où h_0 est centré et décorrélé de (u_t, v_t), et où (u_t, v_t) est une variable aléatoire centrée, décorrélée, stationnaire au sens large et de matrice de covariance

$$\mathbb{E}\left[\begin{bmatrix} u_t \\ v_t \end{bmatrix} \cdot \begin{bmatrix} u_{t'}^T & v_{t'}^T \end{bmatrix}\right] = \begin{bmatrix} Q & S \\ S^T & R \end{bmatrix} \delta_{t,t'}. \tag{A.9}$$

Le couple (h_{t+1}, x_t) est centré et stationnaire, donc h_t et x_t le sont également. Posons $P = \mathbb{E}[h_t h_t^T]$; P vérifie

$$P = FPF^T + Q. \tag{A.10}$$

La fonction de covariance de x_t est donnée par

$$\begin{aligned} r_0 &= \mathbb{E}[x_t^2] \\ &= R + HPH^T; \qquad (A.11) \\ \forall k \in \mathbb{N}^*, r_k &= \mathbb{E}[x_t x_{t-k}] \\ &= HF^{k-1} \underbrace{(FPH^T + S)}_{N}. \qquad (A.12) \end{aligned}$$

Partant d'une covariance $(r_k)_{k\in\mathbb{N}}$, le problème de la réalisation stochastique consiste à construire une « représentation markovienne » minimale de $(x_t)_{t\in\mathbb{N}}$, c'est-à-dire un système d'état (A.8)(A.9), avec F de dimension minimale.

Étape 1

Du fait de la structure de la fonction $(r_k)_{k\in\mathbb{N}^*}$, nous pouvons comme dans le paragraphe A.1 construire une matrice de Hankel

$$\mathcal{R}_\infty = \begin{bmatrix} r_1 & r_2 & r_3 & \dots \\ r_2 & r_3 & & \\ r_3 & & & \\ \vdots & & & \end{bmatrix} \tag{A.13}$$

$$= \begin{bmatrix} H \\ HF \\ HF^2 \\ \vdots \end{bmatrix} \cdot \begin{bmatrix} N & FN & F^2N & \dots \end{bmatrix} \tag{A.14}$$

qui doit bien sûr être rapprochée de la factorisation (A.7). La première étape d'un algorithme de réalisation stochastique consiste donc à déterminer une réalisation (H, F, N) de $(r_k)_{k\in\mathbb{N}^*}$ (unique à une matrice inversible près).

Étape 2

À ce stade, nous disposons de (H, F, N) mais N reste une fonction de P et de S d'après l'équation (A.12), et il reste par ailleurs à identifier Q et R. Cette deuxième étape est plus délicate car le problème doit être résolu sous contraintes de positivité : en effet, les matrices P et $\begin{bmatrix} Q & S \\ S^T & R \end{bmatrix}$ sont des matrices de covariance et doivent donc être semi-définies positives. Si ces contraintes n'étaient pas vérifiées, la solution obtenue n'aurait pas de sens.

Finalement, le problème est le suivant : connaissant (H, F, N, r_0), nous cherchons (P, Q, R, S)

tels que

$$\begin{bmatrix} P & N \\ N^T & r_0 \end{bmatrix} - \begin{bmatrix} F \\ H \end{bmatrix} P \begin{bmatrix} F^T & H^T \end{bmatrix} = \begin{bmatrix} Q & S \\ S^T & R \end{bmatrix}, \tag{A.15}$$

$$P > 0, \tag{A.16}$$

$$\begin{bmatrix} Q & S \\ S^T & R \end{bmatrix} \geq 0. \tag{A.17}$$

On remarque que l'équation (A.15) exprime en fait la covariance de (h_{t+1}, x_t). Puisque (u_t, v_t) est un bruit blanc (une suite de variables aléatoires décorrélées dans le temps), cette covariance vérifie une équation (dite de Ricatti) de même nature que celle vérifiée par P (équation (A.10), qui n'est autre qu'une sous-matrice de l'équation matricielle (A.15)). Le système (A.15) implique que Q, R et S sont uniques une fois P fixée, et donc que l'ensemble des solutions du système contraint (A.15)-(A.17) est paramétré par un ensemble $\mathcal{P}$ de matrices définies positives. Notons enfin que la contrainte sur P devrait être, *a priori*, que P est *semi-définie* positive, mais il se trouve en réalité que toute solution P doit être *définie* positive [Faurre, 1979] (voir le théorème 7 ci-dessous), d'où l'écriture de la contrainte (A.16).

Lemme positif réel, structure de l'ensemble $\mathcal{P}$

Un résultat connu sous le nom de *lemme positif réel* (initialement établi dans le domaine spectral) relie la positivité de la séquence $(r_k)_{k\in\mathbb{N}}$ à l'existence d'au moins une solution au système contraint (A.15)-(A.17). Rappelons que la suite infinie $(r_k)_{k\in\mathbb{N}}$ définit une fonction de covariance si et seulement si la forme de Toeplitz $\sum_{i,j=0}^{m} u_i u_j r_{|j-i|}$ est positive ou nulle pour tout m, c'est-à-dire si et seulement si la matrice de Toeplitz associée est semi-définie positive pour tout m.

Lemme 7 (Lemme positif réel [Faurre, 1976]) *La suite $(r_k)_{k\in\mathbb{N}}$ est une fonction de covariance si et seulement si $\mathcal{P}$ est non vide.*

Nous en venons désormais à la structure de $\mathcal{P}$.

Théorème 7 ([Faurre, 1979]) *L'ensemble $\mathcal{P}$ est un ensemble fermé, convexe, borné et défini positif admettant, pour la relation d'ordre usuelle entre matrices symétriques, un maximum P^* et un minimum P_*.*

Bibliographie

[ntp, 2014] (2014). Network time protocol. http://www.ntp.org/.

[kaf, 2017] (2017). Apache Kafka. https://kafka.apache.org/.

[clo, 2019] (2019). Cloudflare. https://www.cloudflare.com/fr-fr/cdn/.

[log, 2019] (2019). Logentries. https://logentries.com/.

[ren, 2019] (2019). Renater. https://www.renater.fr/.

[ano, 2020] (2020). Anodot. https://www.anodot.com/.

[cac, 2020] (2020). Cacti. https://www.cacti.net/.

[ela, 2020] (2020). Elasticsearch. https://www.elastic.co/fr/elasticsearch/.

[log, 2020] (2020). Loggly. https://www.loggly.com/.

[mun, 2020] (2020). Munin. http://munin-monitoring.org/.

[nag, 2020] (2020). Nagios. https://www.nagios.org/.

[oti, 2020] (2020). Open Transit Internet. https://wholesalefrance.orange.fr/fr/nos-solutions/interconnexion/fixe/open-transit-internet/.

[pac, 2020] (2020). PacketAI. https://packetai.co/.

[rsy, 2020] (2020). `rsyslog`. https://www.rsyslog.com/.

[tcp, 2021] (2021). `tcpdump`. https://www.tcpdump.org/.

[wir, 2021] (2021). Wireshark. https://www.wireshark.org/.

[Aghasaryan et al., 1998] Aghasaryan, A., Fabre, E., Benveniste, A., Boubour, R., and Jard, C. (1998). Fault detection and diagnosis in distributed systems : an approach by partially stochastic Petri nets. *Discrete event dynamic systems*, 8(2) :203–231.

[Aghasaryan et al., 2018] Aghasaryan, A., Kostadinov, D., and Bouzid, M. (2018). Automatic root cause analysis for large-scale dynamic networks. *Data Science Summer School*.

[Agrawal et al., 1994] Agrawal, R., Srikant, R., et al. (1994). Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499.

[Aho and Corasick, 1975] Aho, A. V. and Corasick, M. J. (1975). Efficient string matching : an aid to bibliographic search. *Communications of the ACM*, 18(6) :333–340.

[Aho et al., 1972] Aho, A. V., Garey, M. R., and Ullman, J. D. (1972). The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1(2) :131–137.

[Akhiezer and Kemmer, 1965] Akhiezer, N. I. and Kemmer, N. (1965). *The classical moment problem and some related questions in analysis*, volume 5. Oliver & Boyd Edinburgh.

[Alaeddini and Dogan, 2011] Alaeddini, A. and Dogan, I. (2011). Using Bayesian networks for root cause analysis in statistical process control. *Expert Systems with Applications*, 38(9) :11230–11243.

[Altschul et al., 1990] Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. (1990). Basic local alignment search tool. *Journal of molecular biology*, 215(3) :403–410.

[Aluja-Banet and Nafria, 2003] Aluja-Banet, T. and Nafria, E. (2003). Stability and scalability in decision trees. *Computational Statistics*, 18(3) :505–520.

[Anderson and Moore, 2012] Anderson, B. D. and Moore, J. B. (2012). *Optimal filtering*. Courier Corporation.

[Andersson et al., 2007] Andersson, L., Minei, I., and Thomas, B. (2007). LDP specificatino. *RFC 5036*.

[Arulampalam et al., 2002] Arulampalam, M. S., Maskell, S., Gordon, N., and Clapp, T. (2002). A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Transactions on signal processing*, 50(2) :174–188.

[Baum and Eagon, 1967] Baum, L. E. and Eagon, J. A. (1967). An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology. *Bulletin of the American Mathematical Society*, 73(3) :360–363.

[Baum and Petrie, 1966] Baum, L. E. and Petrie, T. (1966). Statistical inference for probabilistic functions of finite state Markov chains. *The annals of mathematical statistics*, 37(6) :1554–1563.

[Baum et al., 1970] Baum, L. E., Petrie, T., Soules, G., and Weiss, N. (1970). A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The annals of mathematical statistics*, 41(1) :164–171.

[Bengio and Frasconi, 1995] Bengio, Y. and Frasconi, P. (1995). Diffusion of credit in Markovian models. In *Advances in Neural Information Processing Systems*, pages 553–560.

[Bishop, 2006] Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

[Borovkov, 1987] Borovkov, A. A. (1987). *Statistique mathématique*. Editions Mir.

[Bouillard et al., 2018] Bouillard, A., Buob, M.-O., Raynal, M., and Salaün, A. (2018). Log analysis via space-time pattern matching. In *2018 14th International Conference on Network and Service Management (CNSM)*, pages 303–307. IEEE.

[Bouillard et al., 2012] Bouillard, A., Junier, A., and Ronot, B. (2012). Hidden anomaly detection in telecommunication networks. In *2012 8th international conference on network and service management (cnsm) and 2012 workshop on systems virtualiztion management (svm)*, pages 82–90. IEEE.

[Braden et al., 1997] Braden, R., Zhang, L., Berson, S., Herzog, S., and Jamin, S. (1997). Resource ReSerVation Protocol :(RSVP) ; version 1 functional specification.

[Briers et al., 2010] Briers, M., Doucet, A., and Maskell, S. (2010). Smoothing algorithms for state–space models. *Annals of the Institute of Statistical Mathematics*, 62(1) :61.

[Brockett, 2015] Brockett, R. W. (2015). *Finite dimensional linear systems*. SIAM.

[Bruynooghe, 1978] Bruynooghe, M. (1978). Classification ascendante hiérarchique des grands ensembles de données : un algorithme rapide fondé sur la construction des voisinages réductibles. *Cahiers de l'analyse des données*, 3(1) :7–33.

[Burges, 1998] Burges, C. J. (1998). A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2) :121–167.

[Caines, 2018] Caines, P. E. (2018). *Linear stochastic systems*, volume 77. SIAM.

[Carvalho et al., 2010] Carvalho, C. M., Johannes, M. S., Lopes, H. F., Polson, N. G., et al. (2010). Particle learning and smoothing. *Statistical Science*, 25(1) :88–106.

[Chen, 1970] Chen, C.-T. (1970). *Introduction to linear system theory*. Holt, Rinehart and Winston.

[Chen, 1993] Chen, G. (1993). *Approximate Kalman filtering*, volume 2. World Scientific.

[Chen et al., 2004] Chen, M., Zheng, A. X., Lloyd, J., Jordan, M. I., and Brinewer, E. (2004). Failure diagnosis using decision trees. In *International Conference on Autonomic Computing, 2004. Proceedings.*, pages 36–43. IEEE.

[Cheng et al., 2013] Cheng, Y., Izadi, I., and Chen, T. (2013). Pattern matching of alarm flood sequences by a modified Smith-Waterman algorithm. *Chemical engineering research and design*, 91(6) :1085–1094.

[Chui and Chen, 2012] Chui, C. K. and Chen, G. (2012). *Signal processing and systems theory : selected topics*, volume 26. Springer Science & Business Media.

[Chung et al., 2014] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *NIPS 2014 Workshop on Deep Learning*.

[Cormen et al., 2009] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms, 3rd Edition*. MIT Press.

[Cybenko, 1989] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4) :303–314.

[de Jong, 1975] de Jong, L. S. (1975). *Numerical aspects of realization algorithms in linear systems theory*. PhD thesis, Department of Mathematics and Computer Science.

[de Jong, 1978] de Jong, L. S. (1978). Numerical aspects of recursive realization algorithms. *SIAM Journal on Control and optimization*, 16(4) :646–659.

[Dellaert, 2002] Dellaert, F. (2002). The expectation maximization algorithm. Technical report, Georgia Institute of Technology.

[Dempster et al., 1977] Dempster, A. P., Laird, N., and Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society (B)*, 39(1) :1–38.

[Doucet et al., 2001] Doucet, A., de Freitas, N., and Gordon, N. (2001). Sequential Monte Carlo methods in practice. *Information Science and Statistics (Springer New York, 2001)*.

[Dousson and Duong, 1999] Dousson, C. and Duong, T. V. (1999). Discovering chronicles with numerical time constraints from alarm logs for monitoring dynamic systems. In *IJCAI*, volume 99, pages 620–626.

[Dousson et al., 1993] Dousson, C., Gaborit, P., and Ghallab, M. (1993). Situation recognition : Representation and algorithms. In *IJCAI : International Joint Conference on Artificial Intelligence*, pages 166–174.

[Droms, 1997] Droms, R. (1997). Dynamic host configuration protocol.

[Enders, 2004] Enders, W. (2004). *Stationary Time-Series Models*. New York : Wiley.

[Faurre, 1979] Faurre, P. (1979). *Opérateurs rationnels positifs*. Dunod.

[Faurre, 1976] Faurre, P. L. (1976). Stochastic realization algorithms. In *Mathematics in Science and Engineering*, volume 126, pages 1–25. Elsevier.

[Fearnhead et al., 2010] Fearnhead, P., Wyncoll, D., and Tawn, J. (2010). A sequential smoothing algorithm with linear computational cost. *Biometrika*, 97(2) :447–464.

[Forney, 1973] Forney, G. D. (1973). The Viterbi algorithm. *Proceedings of the IEEE*, 61 :268–78.

[François and Aubry, 2018] François, P.-O. and Aubry, E. (2018). Le dessous des cartes - câbles sous-marins : la guerre invisible. https://boutique.arte.tv/detail/le_dessous_des_cartes_cables_sous_marins_guerre_invisible.

[Gauthier, 2020] Gauthier, M. (2020). Le réseau 3G sera-t-il mis hors service avec l'arrivée de la 5G ? https://www.liberation.fr/checknews/2020/07/23/le-reseau-3g-sera-t-il-mis-hors-service-avec-l-arrivee-de-la-5g_1794496.

[Gevers, 2006] Gevers, M. (2006). A personal view of the development of system identification : A 30-year journey through an exciting field. *IEEE Control systems magazine*, 26(6) :93–105.

[Gevers and Wouters, 1978] Gevers, M. and Wouters, W. (1978). An innovations approach to the discrete-time stochastic realization problem. *Journal A*, 19(2) :90–110.

[Goldberg, 2017] Goldberg, Y. (2017). Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10(1) :1–309.

[Gomez-Rodriguez et al., 2012] Gomez-Rodriguez, M., Leskovec, J., and Krause, A. (2012). Inferring networks of diffusion and influence. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 5(4) :1–37.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. (2016). *Deep learning*, volume 1. MIT press Cambridge.

[Granger, 1969] Granger, C. W. (1969). Investigating causal relations by econometric models and cross-spectral methods. *Econometrica : journal of the Econometric Society*, pages 424–438.

[Hammersley, 2013] Hammersley, J. (2013). *Monte Carlo methods*. Springer Science & Business Media.

[Han et al., 2000] Han, J., Pei, J., and Yin, Y. (2000). Mining frequent patterns without candidate generation. *ACM sigmod record*, 29(2) :1–12.

[Harvey, 1990] Harvey, A. C. (1990). *Forecasting, structural time series models and the Kalman filter*. Cambridge university press.

[Hastie et al., 2009] Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The elements of statistical learning : data mining, inference, and prediction*. Springer Science & Business Media.

[Hecht-Nielsen, 1992] Hecht-Nielsen, R. (1992). Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier.

[Ho and Kalman, 1966] Ho, B. and Kalman, R. E. (1966). Effective construction of linear state-variable models from input/output functions. *at-Automatisierungstechnik*, 14(1-12) :545–548.

[Ho and Lee, 1964] Ho, Y. and Lee, R. (1964). A Bayesian approach to problems in stochastic estimation and control. *IEEE transactions on automatic control*, 9(4) :333–339.

[Hornik, 1991] Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2) :251–257.

[Hu et al., 2003] Hu, W., Liao, Y., and Vemuri, V. R. (2003). Robust anomaly detection using support vector machines. In *Proceedings of the international conference on machine learning*, pages 282–289. Citeseer.

[Hume, 1739] Hume, D. (1739). *A treatise of human nature*.

[Hunt and Szymanski, 1977] Hunt, J. W. and Szymanski, T. G. (1977). A fast algorithm for computing longest common subsequences. *Communications of the ACM*, 20(5) :350–353.

[Jaccard, 1912] Jaccard, P. (1912). The distribution of the flora in the alpine zone. 1. *New phytologist*, 11(2) :37–50.

[Jain et al., 1996] Jain, A. K., Mao, J., and Mohiuddin, K. M. (1996). Artificial neural networks : A tutorial. *Computer*, 29(3) :31–44.

[Johannesmeyer et al., 2002] Johannesmeyer, M. C., Singhal, A., and Seborg, D. E. (2002). Pattern matching in historical data. *AIChE journal*, 48(9) :2022–2038.

[Julier et al., 2000] Julier, S., Uhlmann, J., and Durrant-Whyte, H. F. (2000). A new method for the nonlinear transformation of means and covariances in filters and estimators. *IEEE Transactions on automatic control*, 45(3) :477–482.

[Julier and Uhlmann, 2004] Julier, S. J. and Uhlmann, J. K. (2004). Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3) :401–422.

[Julisch, 2003] Julisch, K. (2003). Clustering intrusion detection alarms to support root cause analysis. *ACM transactions on information and system security (TISSEC)*, 6(4) :443–471.

[Kahn and Marshall, 1953] Kahn, H. and Marshall, A. W. (1953). Methods of reducing sample size in Monte Carlo computations. *Journal of the Operations Research Society of America*, 1(5) :263–278.

[Kailath, 1980] Kailath, T. (1980). *Linear systems*, volume 156. Prentice-Hall Englewood Cliffs, NJ.

[Kailath et al., 2000] Kailath, T., Sayed, A. H., and Hassibi, B. (2000). *Linear estimation*. Prentice Hall Information and System Sciences Series. Prentice Hall, Upper Saddle River, New Jersey.

[Kalman, 1960] Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *J. Basic Eng., Trans. ASME, Series D*, 82(1) :35–45.

[Kalman and Bucy, 1961] Kalman, R. E. and Bucy, R. S. (1961). New results in linear filtering and prediction theory. *J. Basic Eng., Trans. ASME, Series D*, 83(3) :95–108.

[Kant, 1781] Kant, I. (1781). *Critique of Pure Reason*.

[Kantas et al., 2015] Kantas, N., Doucet, A., Singh, S. S., Maciejowski, J., and Chopin, N. (2015). On particle methods for parameter estimation in state-space models. *Statist. Sci.*, 30(3) :328–351.

[Koski, 2001] Koski, T. (2001). *Hidden Markov models for bioinformatics*, volume 2. Springer Science & Business Media.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[Latapy et al., 2018] Latapy, M., Viard, T., and Magnien, C. (2018). Stream graphs and link streams for the modeling of interactions over time. *Social Network Analysis and Mining*, 8(1) :61.

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553) :436–444.

[Leibniz, 1710] Leibniz, G. W. (1710). *Theodicity*.

[Lu et al., 2017] Lu, Z., Pu, H., Wang, F., Hu, Z., and Wang, L. (2017). The expressive power of neural networks : A view from the width. In *Advances in neural information processing systems*, pages 6231–6239.

[Lundberg and Lee, 2017] Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Advances in neural information processing systems*, pages 4765–4774.

[Manton and Amblard, 2015] Manton, J. H. and Amblard, P.-O. (2015). A primer on reproducing kernel Hilbert spaces. *Found. Trends Signal Process.*, 8(1–2) :1–126.

[Marilly et al., 2002] Marilly, E., Aghasaryan, A., Betge-Brezetz, S., Martinot, O., and Delegue, G. (2002). Alarm correlation for complex telecommunication networks using neural networks and signal processing. In *IEEE workshop on IP operations and management*, pages 3–7. IEEE.

[McCreight, 1976] McCreight, E. M. (1976). A space-economical suffix tree construction algorithm. *Journal of the ACM (JACM)*, 23(2) :262–272.

[Meinhold and Singpurwalla, 1983] Meinhold, R. J. and Singpurwalla, N. D. (1983). Understanding the Kalman filter. *The American Statistician*, 37(2) :123–127.

[Menegaz et al., 2015] Menegaz, H. M., Ishihara, J. Y., Borges, G. A., and Vargas, A. N. (2015). A systematization of the unscented Kalman filter theory. *IEEE Transactions on automatic control*, 60(10) :2583–2598.

[Mikolov et al., 2011] Mikolov, T., Kombrink, S., Burget, L., Černockỳ, J., and Khudanpur, S. (2011). Extensions of recurrent neural network language model. In *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5528–5531. IEEE.

[Mockapetris, 1987] Mockapetris, P. (1987). Domain names-concepts and facilities. *STD 13, RFC 1034*.

[Molnar, 2020] Molnar, C. (2020). *Interpretable machine learning*. Lulu. com.

[Monin, 2012] Monin, J.-F. (2012). *Understanding formal methods*. Springer Science & Business Media.

[Moy, 1998] Moy, J. (1998). OSPF version 2. *STD 54, RFC 2328*.

[Mozer, 1995] Mozer, M. C. (1995). A focused backpropagation algorithm for temporal. *Backpropagation : Theory, architectures, and applications*, 137.

[Pascanu et al., 2013] Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318.

[Paulsen and Raghupathi, 2016] Paulsen, V. I. and Raghupathi, M. (2016). *An introduction to the theory of reproducing kernel Hilbert spaces*, volume 152. Cambridge University Press.

[Pinkus, 1999] Pinkus, A. (1999). Approximation theory of the MLP model in neural networks. *Acta numerica*, 8(1) :143–195.

[Rabiner, 1989] Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2) :257–286.

[Rabiner and Juang, 1986] Rabiner, L. R. and Juang, B. H. (1986). An introduction to hidden Markov models. *IEEE ASSP Magazine*, pages 4–16.

[Rekhter et al., 1994] Rekhter, Y., Li, T., Hares, S., et al. (1994). A border gateway protocol 4 (BGP-4).

[Revuz, 1992] Revuz, D. (1992). Minimisation of acyclic deterministic automata in linear time. *Theoretical Computer Science*, 92(1) :181–189.

[Ribeiro et al., 2016] Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). "why should I trust you ?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144.

[Robinson and Fallside, 1987] Robinson, A. and Fallside, F. (1987). *The utility driven dynamic error propagation network*. University of Cambridge Department of Engineering Cambridge, MA.

[Rumelhart et al., 1985] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.

[Salaün et al., 2019a] Salaün, A., Bouillard, A., and Buob, M.-O. (2019a). Space-time pattern extraction in alarm logs for network diagnosis. In *MLN 2019 : 2nd IFIP International Conference on Machine Learning for Networking*.

[Salaün et al., 2020] Salaün, A., Bouillard, A., and Buob, M.-O. (2020). Demo abstract : End-to-end root cause analysis of a mobile network. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE.

[Salaün et al., 2019b] Salaün, A., Petetin, Y., and Desbouvries, F. (2019b). Comparing the modeling powers of RNN and HMM. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pages 1496–1499. IEEE.

[Shumway and Stoffer, 1982] Shumway, R. H. and Stoffer, D. (1982). An approach to time series smoothing and forecasting using the EM algorithm. *Journal of Time Series Analysis*, 3(4) :253–264.

[Smith et al., 2008] Smith, R., Japkowicz, N., Dondo, M., and Mason, P. (2008). Using unsupervised learning for network alert correlation. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 308–319. Springer.

[Smith et al., 1981] Smith, T. F., Waterman, M. S., et al. (1981). Identification of common molecular subsequences. *Journal of molecular biology*, 147(1) :195–197.

[Solé et al., 2017] Solé, M., Muntés-Mulero, V., Rana, A. I., and Estrada, G. (2017). Survey on models and techniques for root-cause analysis. *arXiv preprint arXiv :1701.08546*.

[Sorsa and Koivo, 1993] Sorsa, T. and Koivo, H. N. (1993). Application of artificial neural networks in process fault diagnosis. *Automatica*, 29(4) :843–849.

[Srikant and Agrawal, 1996] Srikant, R. and Agrawal, R. (1996). Mining sequential patterns : Generalizations and performance improvements. In *International Conference on Extending Database Technology*, pages 1–17. Springer.

[Subias et al., 2014] Subias, A., Travé-Massuyès, L., and Le Corronc, E. (2014). Learning chronicles signing multiple scenario instances. *IFAC Proceedings Volumes*, 47(3) :10397–10402.

[Tchebychev, 1867] Tchebychev, P. L. (1867). Des valeurs moyennes. *J. Math. Pures Appl*, 12(2) :177–184.

[Thompson, 1968] Thompson, K. (1968). Programming techniques : Regular expression search algorithm. *Communications of the ACM*, 11(6) :419–422.

[Ukkonen, 1995] Ukkonen, E. (1995). On-line construction of suffix trees. *Algorithmica*, 14(3) :249–260.

[Van Lunteren, 2006] Van Lunteren, J. (2006). High-performance pattern-matching for intrusion detection. In *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pages 1–13. Citeseer.

[Vapnik, 2013] Vapnik, V. (2013). *The nature of statistical learning theory*. Springer science & business media.

[Viard et al., 2018] Viard, T., Fournier-S'Niehotta, R., Magnien, C., and Latapy, M. (2018). Discovering patterns of interest in IP traffic using cliques in bipartite link streams. In *International Workshop on Complex Networks*, pages 233–241. Springer.

[Viterbi, 1967] Viterbi, A. J. (1967). Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory*, 13 :260–69.

[Wang et al., 2020] Wang, S., Cao, J., and Yu, P. S. (2020). Deep learning for spatio-temporal data mining : A survey. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1.

[Wasserman, 2013] Wasserman, L. (2013). *All of statistics : a concise course in statistical inference*. Springer Science & Business Media.

[Weidl et al., 2005] Weidl, G., Madsen, A. L., and Israelson, S. (2005). Applications of object-oriented Bayesian networks for condition monitoring, root cause analysis and decision support on operation of complex continuous processes. *Computers & chemical engineering*, 29(9) :1996–2009.

[Werbos, 1990] Werbos, P. J. (1990). Backpropagation through time : what it does and how to do it. *Proceedings of the IEEE*, 78(10) :1550–1560.

[Wu, 1983] Wu, C. J. (1983). On the convergence properties of the EM algorithm. *The Annals of statistics*, pages 95–103.

[Zaki, 2000] Zaki, M. J. (2000). Scalable algorithms for association mining. *IEEE transactions on knowledge and data engineering*, 12(3) :372–390.

[Zaki, 2001] Zaki, M. J. (2001). SPADE : An efficient algorithm for mining frequent sequences. *Machine learning*, 42(1-2) :31–60.

[Zhang et al., 2019] Zhang, C., Song, D., Chen, Y., Feng, X., Lumezanu, C., Cheng, W., Ni, J., Zong, B., Chen, H., and Chawla, N. V. (2019). A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1409–1416.

ECOLE
DOCTORALE
DE MATHEMATIQUES
HADAMARD

Titre : Prédiction d'alarmes dans les réseaux via la recherche de motifs spatio-temporels et l'apprentissage automatique

Mots clés : analyse de logs, algorithmes en ligne, recherche de motifs, modèles de Markov cachés, réseaux de neurones récurrents

Résumé : Les réseaux de télécommunication prennent aujourd'hui une place prépondérante dans notre monde. Ils permettent en effet de partager des informations en masse et à l'échelle planétaire. Toutefois, il s'agit de systèmes complexes, en taille comme en diversité technologique. Cela rend d'autant plus complexe leur maintenance et leur réparation. Afin de limiter l'influence négative de ces dernières, des outils doivent être développés pour détecter une panne dès qu'elle a lieu, en analyser les causes afin de la résoudre efficacement, voire prédire la-dite panne pour *prévenir plutôt que guérir*. Dans cette thèse, nous nous intéressons principalement à ces deux derniers problèmes. Pour cela, nous disposons de fichiers, appelés logs d'alarmes, recensant l'ensemble des alarmes émises par le système. Cependant, ces fichiers sont généralement verbeux et bruités: l'administrateur à la charge d'un réseau doit disposer d'outils capables d'isoler et manipuler de façon interprétable les liens de causalité au sein d'un log. Dans un premier temps, nous construisons en ligne une structure, appelée DIG-DAG, qui stocke toutes les chaînes de causalité impliquant les événements d'un log. Nous proposons ensuite un système de requête pour exploiter cette structure. Enfin, nous appliquons cette approche dans le cadre de l'analyse de causes racines. Dans un second temps, nous abordons une approche générative pour la prédiction de données. En particulier, nous comparons deux modèles classiques pour cette tâche: les réseaux de neurones récurrents d'une part et les modèles de Markov cachés d'autre part. En effet, dans leurs communautés respectives, ces deux modèles font office d'état de l'art. Ici, nous comparons analytiquement leur expressivité grâce à un modèle probabiliste, appelé GUM, unifiant ces deux modèles.

Title : Alarm prediction in networks via space-time pattern matching and machine learning

Keywords : log analysis, online algorithms, pattern matching, hidden Markov models, recurrent neural networks

Abstract : Nowadays, telecommunication networks occupy a predominant place in our world. Indeed, they allow to share worldwide a huge amount of information. Networks are however complex systems, both in size and technological diversity. Therefore, it makes their management and repair more difficult. In order to limit the negative impact of such failures, some tools have to be developed to detect a failure whenever as soons as it occurs, analyse its root causes to solve it efficiently, or even predict this failure to prevent it rather than cure it. In this thesis, we mainly focus on these last two problems. To do so, we use files, called alarm logs, storing all the alarms issued by the system. However, these files are generally noisy and verbose: an operator managing a network needs tools able to extract and handle in an interpretable manner the causal relationships within a log. First, we build online a structure, called DIG-DAG, that stores all the potential causal relationships involving the events of a log. We then propose a query system to exploit this structure. Finally, we apply this approach in the context of root cause analysis. Second, we discuss a generative approach for times series prediction. In particular, we compare two well-known models for this task: recurrent neural nets on the one hand, hidden Markov models on the other hand. Indeed, in their respective communities, these two models are state of the art. Here, we compare analytically their expressivity by encompassing them into a probabilistic model, called GUM.

Institut Polytechnique de Paris
91120 Palaiseau, France